Key Design Features

- Synthesizable, technology independent VHDL IP Core
- Video overlays on 24-bit RGB or YCbCr 4:4:4 video
- Supports all video resolutions up to 2^{16} x 2^{16} pixels
- Supports any number of input video streams or video overlays (by cascading modules together in series)
- Programmable video-overlay position and size
- Per pixel 8-bit alpha transparency
- Choice of ROP commands including AND, OR and XOR
- Simple input and output interfaces
- No complex programming required
- Easily integrates with all Zipcores Video IP
- Small implementation size
- Supports FPGA clock rates in excess of 250 MHz[1]

Applications

- Digital TV and home-media solutions
- Broadcast TV and film production
- Digital-video special effects
- Multiple video windows
- Animated video windows
- Picture-in-Picture (PiP) applications
- Instrumentation and monitoring
- Network and Tactical operations centres
- CCTV and security camera systems

Block Diagram

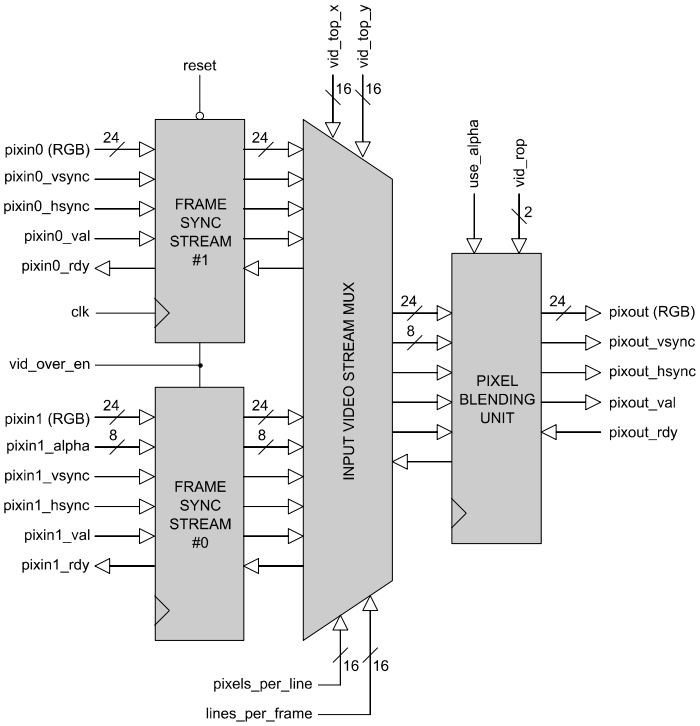

Figure 1: Digital Video Overlay architecture

Pin-out Description

Pin name	I/O	Description	Active state
clk	in	Synchronous clock	rising edge
reset	in	Asynchronous reset	low
vid_over_en	in	Enable/disable video overlay (When disabled, the background video passes though unchanged)	high
vid_rop [1:0]	in	Raster Operation (Performs bitwise operation between background video and overlay video) 0: NOP, 1: AND, 2: OR, 3: XOR	data
vid_top_x [15:0]	in	Top-left x position of video-overlay	data
vid_top_y [15:0]	in	Top-left y-position of video-overlay	data
pixels_per_line [15:0]	in	Number of pixels per line in the overlay video	data
lines_per_frame [15:0]	in	Number of lines per frame in the overlay video	data

1 Xilinx® Virtex 6 FPGA used as a benchmark

Copyright © 2014 www.zipcores.com

Pin-out Description cont ...

Pin name	I/O	Description	Active state
pixin0 [23:0]	in	24-bit RGB source pixel in (background video)	data
pixin0_vsync	in	Vertical sync in (Coincident with first pixel of frame)	high
pixin0_hsync	in	Horizontal sync in (Coincident with first pixel of line)	high
pixin0_val	in	Input pixel valid	high
pixin0_rdy	out	Ready to accept input pixel (handshake signal)	high
pixin1 [23:0]	in	24-bit RGB source pixel in (overlay video)	data
pixin1_alpha[7:0]	in	Overlay pixel transparency 0: Fully transparent 1: Fully opaque	data
pixin1_vsync	in	Vertical sync in (Coincident with first pixel of frame)	high
pixin1_hsync	in	Horizontal sync in (Coincident with first pixel of line)	high
pixin1_val	in	Input pixel valid	high
pixin1_rdy	out	Ready to accept input pixel (handshake signal)	high
pixout [23:0]	out	24-bit RGB output pixel	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	high

General Description

VID_OVERLAY is a highly versatile video multiplexer that allows one video stream to be inserted over another. By cascading a series of video overlay modules together, any number of video sources may be multiplexed together. The module supports input video streams of any resolution or aspect ratio up to 2^{16} x 2^{16} pixels in size. Video overlay parameters may be changed on a frame-by-frame basis to dynamically change the size and position of the video overlay.

Pixels and syncs flow in and out of the video overlay module in accordance with the valid-ready pipeline protocol. Pixels and syncs are sampled at the module inputs on a rising clock-edge when '*val*' is high and '*rdy*' is high. Likewise, pixels and syncs are transferred out of the module on a rising clock-edge when '*val*' is high and '*rdy*' is high. The pipeline protocol allows both input and output interfaces to be stalled independently.

The dimensions of the output video are controlled entirely by the background video stream (*pixin0*) . This means that if the source video input at stream '0' is 1024x768 pixels then the output video will also be 1024x768 pixels in resolution. The video overlay enters the module via stream '1' and must be smaller or equal in size to the background video stream.

In addition, the overlay module supports a number of blending operations including an 8-bit alpha channel and bitwise AND, OR and XOR functions. Figure 1 shows the architecture of the digital video overlay module in more detail.

Input Frame Sync units

Input pixels for both the background video (*pixin0*) and the overlay video (*pixin1*) enter via two separate frame sync modules. The purpose of each module is to find the first 'vsync' of both streams in order synchronize the output frame correctly. If the vsync of the background video is found first, then further pixels in the background image are held off until the the corresponding vsync in the overlay stream is found. Likewise, if the overlay vsync is found first then further overlay pixels are held off until the background vsync is found. Once both vsyncs are found then normal operation begins with the background video stream controlling the flow of pixels into the module.

If at any point a system reset occurs, then the module will resynchronize to the next start of frame for both the background and overlay streams before normal operation continues.

Input Video Multiplexer

The video multiplexer is responsible for controlling the flow of pixels into the video overlay module. It's main function is to detect whether the current pixel to be displayed lies within the overlay region defined by the parameters *vid_top_x*, *vid_top_y*, *pixels_per_line* and *lines_per_frame*. If the current pixel lies outside the overlay region, the input pixel from the background video passes though unchanged. If the pixel lies inside the overlay region, then pixels are processed in the pixel blending unit.

The video overlay position and size may be updated as and when required. If these parameters are not static, then it is desirable that these parameters be updated simultaneously and once per frame. To be extra sure the *reset* signal maybe toggled for at least one clock cycle after changing parameters. This will flush all pixels from the pipeline and cause the input streams to re-sync.

Figure 2 shows the relationship between the background video display area and the defined video overlay window. It is important that the overlay parameters exactly match the size of the video overlay. In addition, the whole video overlay window must lie within the dimensions of the background video - otherwise image corruption will result.

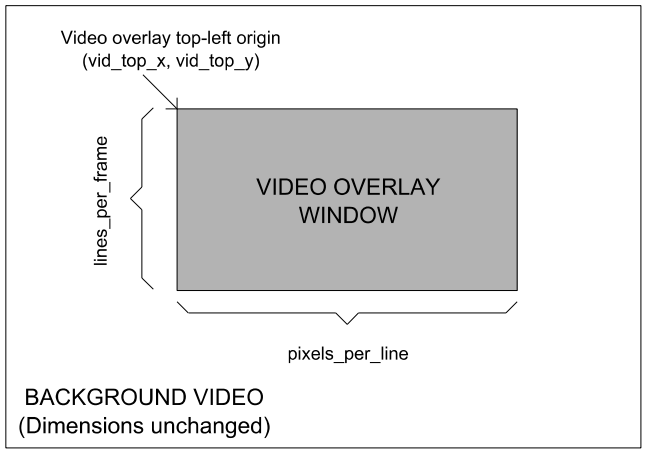

Figure 2: Video overlay window positioning and size

Pixel blending unit

The pixel blending unit generates the output pixel to be displayed. The appearance of the pixel depends on whether the pixel lies inside the video overlay window and also the chosen blending parameters. Figure 3 demonstrates the action of the blender graphically.

Image (a) shows the background video without overlay. This represents the output video when the signal *vid_over_en* is set to logic '0'. In this state, the video overlay is ignored and only the background video is displayed.

Image (b) shows normal overlay operation without blending. In this case, the signal *vid_over_en* is set to '1', the transparency is fixed and set to '0xFF' and the ROP command is set to '0'.

Image (c) shows the same overlay with the transparency set to 50% or '0x7F'. Again, the ROP command is set to '0'.

Image (d) demonstrates per-pixel alpha-blending. The alpha value may be changed on a per-pixel basis. In this example, the transparency has been increased from left to right resulting in a graduated effect.

Images (e), (f) and (g) demonstrate the bitwise AND, OR and XOR operations between the background and overlay video streams. Logical operations are performed independently on the Red, Green and Blue channels for each pixel. The table below summaries the basic blending commands.

Overlay enable (1-bit)	***Alpha (8-bits)***	***ROP command (2-bits)***	***Operation***
0x0	0xXX	0xX	Overlay disabled
0x1	0x7F 0x55	0x0 0x0	50% transparency 66% transparency etc ..
0x1	0xXX	0x1	Bitwise AND
0x1	0xXX	0x2	Bitwise OR
0x1	0xXX	0x3	Bitwise XOR

Alpha-blending is useful if the user wishes to 'fade-in' or 'fade-out' the overlay video in the main display. Arbitrary parts of the overlay video may also be bought into and out of view by controlling the alpha value on a per-pixel basis.

Likewise, the bitwise ROP commands are useful for various special effects such as the generation of sprites, masks and chroma-keying.

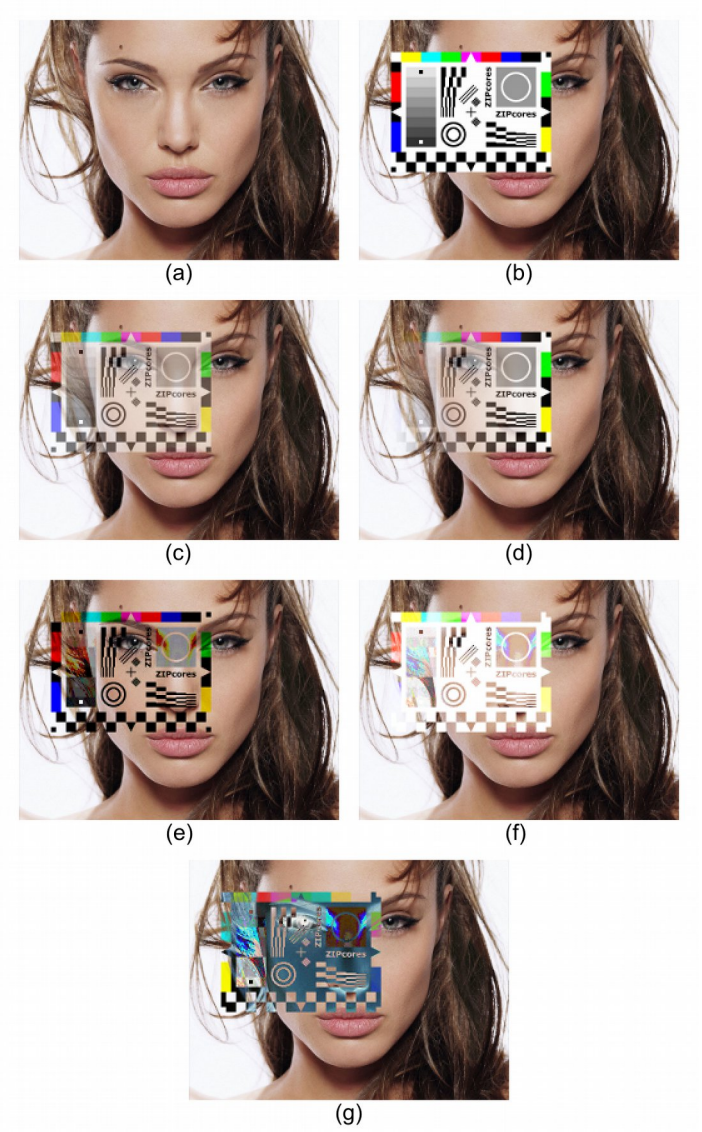

Figure 3: (a) Overlay disabled, (b) No blending (c) 50% alpha, (d) Graduated alpha, (e) Bitwise AND, (f) Bitwise OR, (g) Bitwise XOR

Functional Timing

All RGB input and output pixels are sampled according to the valid-ready pipeline protocol[2]. Figure 4 shows the signalling at the pixin0 input (the background video source) at the start of a new frame.

The first line of a new frame begins with *pixin0_vsync* and *pixin0_hsync* asserted high together with the first pixel. It is important to note that input pixels and syncs are only sampled on a rising clock-edge when *pixin0_val* and *pixin0_rdy* are both high. If this protocol is not observed, then pixels will be lost and the resulting output video will be corrupted.

2 See application note: app_note_zc001.pdf on the Zipcores website for more examples of the valid-ready pipeline protocol

As an example, the waveform shows what happens when *pixin0_rdy* is de-asserted. In this case, the pipeline is stalled and the upstream interface must hold-off before further pixels are sampled.

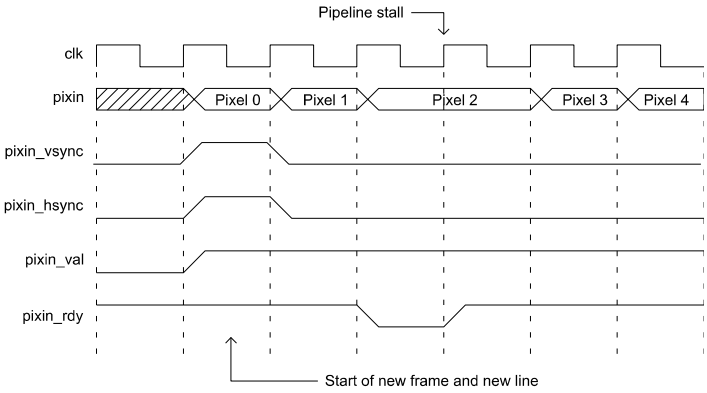

Figure 4: First line of a new frame

Figure 4 timing waveforms are equally valid for the *pixin1* (overlay video) interface. Exactly the same valid-ready signalling is used. However, as the video overlay will generally be smaller in size than the background video, then it will be expected that the *pixin1* interface will be stalled for a greater proportion of the time (signified by *pixin1_rdy* being de-asserted). The *pixin1* interface also differs in that it supports an 8-bit alpha channel *pixin1_alpha* which specifies the transparency of the overlaid pixel.

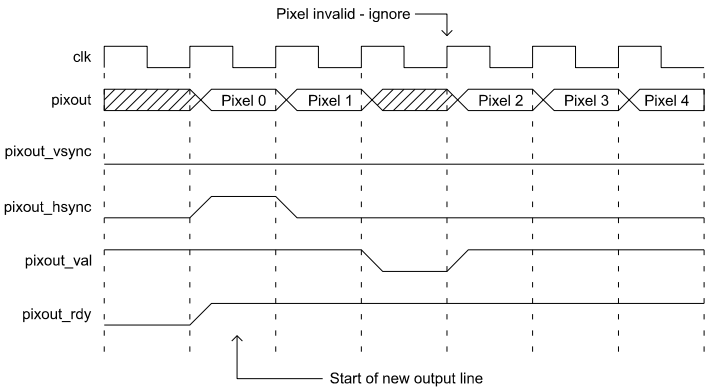

Figure 5: Video overlay output showing invalid pixel

Figure 5 shows the signalling at the output of the video overlay module. 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.

All other parameters used by the video overlay module - including the video window position, size and ROP command are sampled continuously on the rising edge of the system clock. In the following clock cycle, these parameters will be active and be ready for use.

Note: when the signal *vid_over_en* is set to '0' then the overlay video is disabled. In this state, only the background video pixels are processed. The *pixin1* ports are effectively ignored with the signal *pixin1_rdy* held low.

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_bgnd.txt	Background video source text file
video_in_over.txt	Overlay video source text file
char_file_reader.vhd	Character buffer input file reader
pipeline_reg.vhd	Pipeline register component
pipeline_shovel.vhd	Pipeline 'shovel' register component
vid_sync_sof.vhd	Start of frame synchronizer
vid_overlay_reader.vhd	Source video file reader
vid_overlay.vhd	Top-level component
vid_overlay_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. pipeline_reg.vhd
2. pipeline_shovel.vhd
3. vid_sync_sof.vhd
4. vid_blend.vhd
5. vid_overlay.vhd
6. vid_overlay_reader.vhd
7. vid_overlay_bench.vhd

The VHDL testbench instantiates the VID_OVERLAY component and the user may modify the generic parameters as required. In the example testbench provided, a 640x480 (VGA) image is used as the background source video and a 320x240 Zipcores testcard is used as the video overlay.

The source video for the simulation is generated by the video file-reader component. There are two instantiations of this component - one for the background video and one for the overlay video. The simulation requires two text files to be placed in the top-level simulation directory. These files are called *video_in_bgnd.txt* and *video_in_over.txt* and they contain the source pixels and syncs for the test. Both files follow a simple format which define the state of signals: *pixin_val*, *pixin_vsync*, *pixin_hsync, pixin* and *pixin_alpha* on a clock-by-clock basis.

An example file might be the following:

```
1  1  1  00 11 22 00  # pixel 0 line 0 (start of frame)
1  0  0  33 44 55 10  # pixel 1 line 0
0  0  0  00 00 00 00  # don't care!
1  0  0  66 77 88 20  # pixel 2 line 0
.
.
1  0  1  00 11 22 30  # pixel 0 line 1 (start of line)
1  0  0  33 44 55 40  # pixel 1 line 1 etc ..
```

In this example, the first line of of the file asserts the input signals pixin_val = 1, pixin_vsync = 1, pixin_hsync = 1, pixin = 0x001122 and pixin_alpha = 0x00.

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 the input video text files.

The resulting output video from the test is the same as that for Figure 4(d). The output demonstrates a video overlay with a graduated alpha channel positioned (64,64) pixels from the top-left corner of the background video.

Example Video-overlay outputs

Figure 6 is the output of six video-overlay modules cascaded in series - each module configured to display a separate overlay. The result is a multi-tiled display - each one with a separate video source. In this particular example, alpha blending support has been disabled.

Figure 6: Multi-tiled display

The next example (Figure 7) demonstrates the versatility of the video overlay module. In this particular instance, five separate video sources have been multiplexed together. The logo in the top-right corner is alpha blended, as is the bottom section containing the text. Notice that any size of video overlay is supported in any position relative to the main background image. If desired, the position and size of the video overlays may be changed on a frame-by-frame basis for a fully dynamic display.

Figure 7: Multi-format display

Synthesis

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

- vid_overlay.vhd
  - vid_blend.vhd
    - pipeline_reg.vhd
  - vid_sync_sof.vhd
    - pipeline_reg.vhd
  - pipeline_reg.vhd
  - pipeline_shovel.vhd

The VHDL core is designed to be technology independent. However, as a benchmark, synthesis results have been provided for the Xilinx® Virtex6 and Spartan6 FPGA devices. Synthesis results for other FPGAs and technologies can be provided on request.

Note that the generic parameter *use_alpha* will effect the number of hardware multipliers used in the design. If *use_alpha* is set to 'false' then the alpha blending hardware is disabled - resulting in a saving in multiplier resources.

If any of the dynamic parameters such as the ROP command or overlay position are fixed, then these signals should be tied to a constant value at the top-level to allow internal optimizations to be made during synthesis.

Trial synthesis results are shown in the following tables. The generic parameter *use_alpha* has been set to *true*.

Resource usage is specified after Place and Route.

VIRTEX 6

Resource type	***Quantity used***
Slice register	932
Slice LUT	1156
Block RAM	3
DSP48	12
Occupied Slices	401
Clock frequency (approx)	250 MHz

SPARTAN 6

Resource type	***Quantity used***
Slice register	492
Slice LUT	313
Block RAM	0
DSP48	6
Occupied Slices	159
Clock frequency (approx)	170 MHz