Key Design Features

- Synthesizable, technology independent IP Core for FPGA, ASIC or SoC
- Supplied as human readable VHDL (or Verilog) source code
- Output supports full flow control permitting output pixels to be stalled (or even whole frames if necessary)
- Supports *any* video resolution[1]
- Support for RGB or YCbCr pixel formats
- Includes frame skip and frame repeat functionality to compensate for different input and output frame rates
- Generic 128-bit external memory interface with configurable burst size
- Linear memory bursts minimise page-breaks in synchronous memory architectures
- Ideal for interfacing to all types of memory such as SRAM, SDRAM, DDR, DDR2, DDR3, DDR4 etc.
- Supports 300 MHz+ operation on basic FPGA devices[2]

Applications

- Buffering video frames in external memory
- Real-time digital video applications
- Video genlock applications
- Adapting to different pixel-clock rates and frame rates
- Essential component in video processing pipelines

Generic Parameters

Generic name	*Description*	*Type*	*Valid range*
bits_per_pixel (bbp)	Input video bits per pixel	integer	16, 24 or 32
mem_start_addr	Start address in memory of frame buffer (128-bit aligned)	integer	≥ 0
mem_burst_size	Size of memory read / write burst (in 128-bit words)	integer	≥ 2
mem_frame_repeat	Enable / disable frame repeat mode	boolean	True/False

Block Diagram

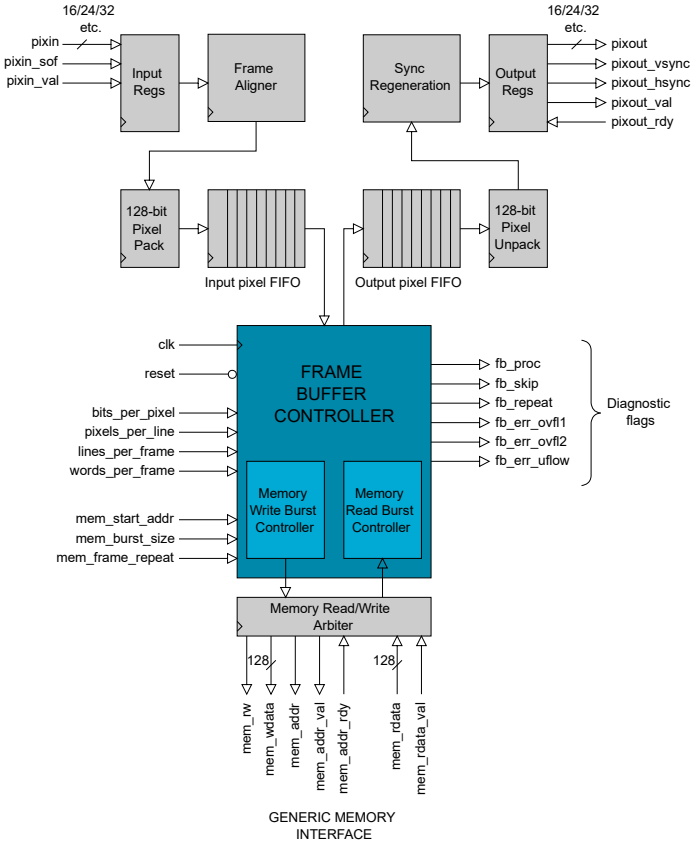

Figure 1: Video Frame Buffer architecture

Pin-out Description

SYSTEM SIGNALS

Pin name	*I/O*	*Description*	*Active state*
clk	in	Synchronous system clock	rising edge
reset	in	Asynchronous system reset	low
fb_proc	out	Frame processed strobe	high
fb_skip	out	Frame skip strobe	high-pulse
fb_repeat	out	Frame repeat strobe (when repeat enabled)	high-pulse
fb_err_ovfl1	out	Input FIFO overflow error	high
fb_err_ovfl2	out	Output FIFO overflow error	high
fb_err_uflow	out	Output pixel underflow flag	high

1 External memory permitting
2 Xilinx® 7-series used as a benchmark

INPUT VIDEO INTERFACE

Pin name	I/O	Description	Active state
pixin [bits_per_pixel - 1:0]	in	Input pixel	data
pixin_sof	in	Start of frame flag (coincident with first pixel in frame)	high
pixin_val	in	Input pixel valid	high

PROGRAMMABLE INPUT VIDEO PARAMETERS

Pin name	I/O	Description	Active state
pixels_per_line (ppl) [15:0]	in	Number of pixels in each line of input video	data
lines_per_frame (lpf) [15:0]	in	Number of lines in each frame of input video	data
words_per_frame [31:0]	in	Size of one frame in 128-bit words (ppl * lpf * bbp) / 128	data

OUTPUT VIDEO INTERFACE

Pin name	I/O	Description	Active state
pixout [bits_per_pixel - 1:0]	out	Output pixel	data
pixout_vsync	out	Vertical sync flag (coincident with first pixel in frame)	high
pixout_hsync	out	Horizontal sync flag (coincident with first pixel in line)	high
pixout_val	out	Output pixel valid	high
pixout_rdy	in	Ready to accept output pixel (handshake signal)	high

GENERIC 128-BIT MEMORY INTERFACE

Pin name	I/O	Description	Active state
mem_rw	out	Memory read / write flag	0: write 1: read
mem_wdata [127:0]	out	Memory write data	data
mem_addr [31:0]	out	Memory read / write address	data
mem_addr_val	out	Memory request valid	high
mem_addr_rdy	in	Ready to accept memory request (handshake signal)	high
mem_rdata [127:0]	in	Memory read data	data
mem_rdata_val	in	Memory read data valid	high

General Description

The VID_FRAME_BUFFER (VFB) IP Core is a high-speed multi-format video frame buffer that samples an input video stream and buffers it in an external memory. The VFB is capable of very high-speed operation - achieving over 300 MHz on standard FPGA platforms.

The VFB will automatically adapt to different input and output frame rates. If the input frame rate is too high, then the VFB will drop or 'skip' an input frame. Likewise, if the output frame rate is higher than the input frame rate, then frames will be repeated[3]. The result is a system that seamlessly adapts to the different frame rates at the input and output of the VFB.

The memory port is a generic 128-bit read/write interface that may be connected to a wide variety of memory types and memory controllers. Memory read/write requests are sent as a sequential linear burst that is optimized for transfers over synchronous memory.

By using a series of VFB IP Cores in parallel, multiple video-sources may be synchronized together. Figure 1. shows the architecture of the Video Frame Buffer in more detail.

Input video interface

The VFB supports any input pixel format as long as the pixels are aligned to a 16, 24 or 32-bit word boundary. Input pixels are sampled on the rising-edge of the system clock when *pixin_val* is high. The signal *pixin_sof* is an active high flag that is coincident with the first pixel of the input frame.

Note that the input video interface is free running and non-stallable. If the input frame rate is too high for the available memory bandwidth, then input frames will be dropped.

Output video interface

Pixels flow out of the VFB in accordance with the valid-ready pipeline protocol. This protocol is used by all Zipcores video IP, and allows for simple connectivity between modules.

Output pixels and syncs are transferred out of the VFB on the rising edge of the system clock when *pixin_val* and *pixin_rdy* are both high. In addition, the output may be stalled, allowing pixels (or even whole frames) to be held back by asserting *pixout_rdy* low. In order to identify the boundary between frames and lines, the sync signals *pixout_vsync* and *pixout_hsync* are provided. The vsync signal is asserted with the first output pixel of a frame and the hsync signal is asserted with the first output pixel of a line.

Generic memory interface

The memory interface is a generic single-ported 128-bit read/write type that may be connected to a wide variety of memories and memory controllers.

Each memory request is sent using the valid-ready protocol. A request is transferred on a rising clock edge when *mem_addr_val* and *mem_addr_rdy* are asserted high. If the request is a write then the flag *mem_rw* is asserted low. For a memory read, then the *mem_rw* flag is asserted high. The *mem_addr* signal is common to both read and write requests.

3 Assuming frame-repeat mode is enabled

Requests are sent as a sequential linear burst with the number of words in each burst being controlled by the generic parameter *mem_burst_size*.

The burst size controls the number of sequential read or write requests. Setting a larger burst size will increase the number sequential accesses to memory and potentially lower the number of page-breaks. Conversely, making the burst size too large may starve the next read or write request of memory bandwidth. For this reason, care should be taken when selecting this parameter.

The parameter *words_per_frame* defines the size of one complete frame of input video in 128-bit words. The parameter *mem_frame_repeat* determines whether video frames should be repeated if the output frame rate is higher than the input frame rate. Finally, the parameter *mem_start_addr* defines where frame-buffer should start in physical memory. The memory must be large enough to support 4 complete frames of input video. This is shown in figure 2 as a system memory map.

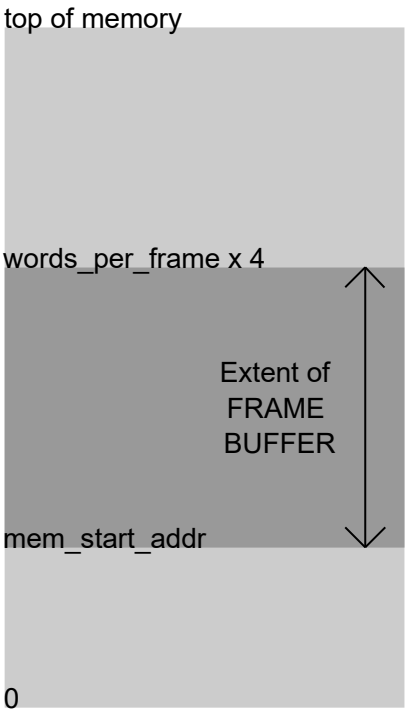

Figure 2: System memory map (128-bit word aligned)

System flags and diagnostic signals

The *fb_skip* flag is an active high strobe that pulses high every time an input frame is dropped. This signal shows activity when the input frame rate is higher than the output frame rate. Conversely, the *fb_repeat* flag pulses high every time an output frame is repeated. This signal will be active when the output frame rate is higher than the input frame rate. The signal *fb_proc* is pulsed high every time an input frame is processed. A combination of all three flags may be used to provide real-time information about the input and output video stream. Figure 3 shows the relationship between the output frames and frame repeat/skip flags.

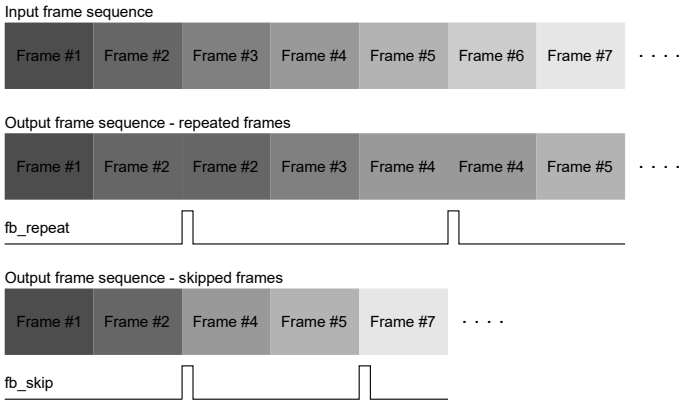

Figure 3: Frame repeat and frame skip flags

In order to maintain a steady video output display, the designer should aim for a well balanced system where the incidence of frame skip and frame repeat is reduced. The optimum system is where the input frame rate and output frame rate are the same or evenly matched.

The most important diagnostic flags to take note of are the signals *fb_err_ovfl1*, *fb_err_ovfl2* and *fb_err_uflow*. The signal *fb_err_ovfl1* indicates that the input FIFOs have overflowed. An input FIFO overflow condition occurs when the input pixel rate is too high. The signal *fb_err_ovfl2* indicates that the output read FIFOs have overflowed[4]. Finally, the *fb_err_uflow* flag is asserted high if there is a dropout of valid output pixels. This is not necessarily an error, but it could indicate a system with insufficient memory read bandwidth.

The only way to recover from an error condition is to assert a system reset. On reset, the VFB will resynchronize to the next input frame and operation will continue as normal.

Practical system considerations

(a) Internally, the VFB is 128-bit word aligned. This means that the size of a single video frame must be divisible by an integer number of 128-bit words. In particular, the following calculation must result in a whole number:

$$\text{words\_per\_frame} = \frac{(\text{pixels\_per\_line} * \text{lines\_per\_frame} * \text{bits\_per\_pixel})}{128}$$

(b) As the memory interface divides each frame into discrete bursts of 128-bit words, the size of a single video frame must be divisible by the memory burst size. Likewise, the following calculation must result in a whole number:

$$\text{bursts\_per\_frame} = \frac{\text{words\_per\_frame}}{\text{mem\_burst\_size}}$$

4 See cases (c) and (f) - Practical system considerations

For common video resolutions, the parameters *words_per_frame* and *mem_burst_size* generally come out as integer numbers. However, for more obscure user-defined video modes, the input video resolution or burst size may need to be adjusted to give integer values.

(c) There comes a point when the input pixel data rate becomes too high for the VFB to tolerate and the input pixel FIFOs overflow. When this happens, even the dropping of individual input frames will not work, as the instantaneous pixel-rate exceeds the maximum bandwidth available. Assuming an 'ideal', non-stalling memory interface where the bandwidth is shared equally between reads and writes, then the minimum system clock frequency required for a given input pixel clock frequency is given by:

$$\text{system\_clock\_frequency} \geq \text{pixel\_clock\_frequency} * (\text{bits\_per\_pixel}/128) * 2$$

As an example, consider a 65 MHz input pixel clock at 24-bits/pixel. The minimum system clock frequency allowed to avoid internal overflow would be: 65*(24/128)*2 = 24.375 MHz. In practice, however, a higher system clock-frequency is often required to compensate for inefficiencies in the memory interface. For instance, due to page-breaks and auto-refresh etc.

(d) In order to minimize the performance bottleneck at the memory interface, the external memory should be clocked at the system clock frequency or better.

$$\text{memory\_clock\_frequency} \geq \text{system\_clock\_frequency}$$

(e) The external memory should be large enough to accommodate up to 4 frames of video. The size in 128-bit words is given by:

$$\text{memory\_size (128-bit)} \geq \frac{(\text{pixels\_per\_line} * \text{lines\_per\_frame} * \text{bits\_per\_pixel} * 4)}{128}$$

For example, consider an XGA (1024x768) input source at 16-bits/pixel. In this case, a minimum memory size of: 1024x768x16x4/128 = 384k x 128-bit would be required. A 1M x 128-bit memory or greater would be a good choice in this instance.

(f) The internal FIFOs have enough buffering to accommodate 7 'in-flight' read memory bursts for a maximum burst size of 64. For this reason, the memory read latency must not exceed 448 system clock cycles. If a very high memory read latency is expected, then please contact Zipcores and the amount of internal buffering can be adjusted accordingly.

Functional Timing

Input video interface

Figure 4 shows the signalling at the input to the VFB. The input pixel and the sof flag are sampled on the rising edge of *clk* when *pixin_val* is high. When *pixin_val* is de-asserted then the input pixel is ignored.

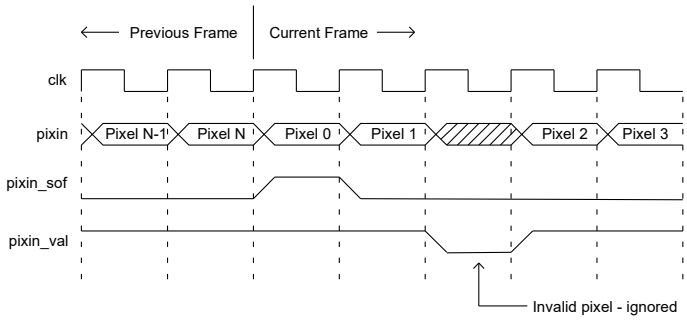

Figure 4: Input video interface timing

Output video interface

Output pixels and syncs are transferred out of the VFB on the rising clock-edge of *clk* when *pixin_val* and *pixin_rdy* are both high. If *pixin_rdy* is held low, then the output is stalled and the frame-buffer will buffer input pixels (or whole frames) until *pixin_rdy* is asserted high again. Figure 5 shows the output video timing at the start of a new output frame. Both *pixin_vsync* and *pixin_hsync* are asserted high with the first pixel of a new frame.

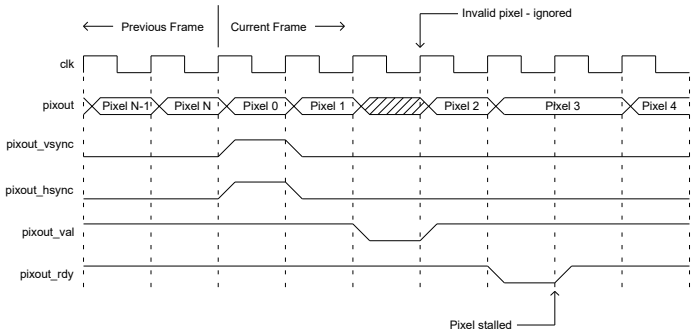

Figure 5: Output video interface timing – start of new output frame

Figure 6 demonstrates the timing at the start of a new line. A new line begins with *pixin_hsync* coincident with the first pixel. The signal *pixin_vsync* is held low.

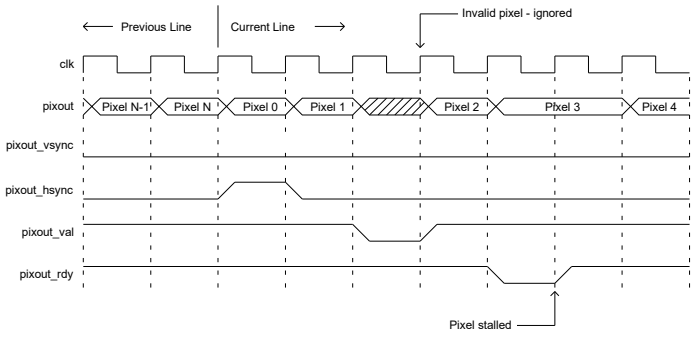

Figure 6: Output video interface timing - start of new output line

Generic 128-bit memory interface

Figure 7 shows a series of write bursts to memory. In this particular example, the parameter *mem_burst_size* has been set to 4[5]. Each memory burst is a block write of 4 words. The addresses are guaranteed to be sequential within a burst. Between bursts, the *mem_addr_valid* signal is de-asserted for one cycle.

At any point during the write transfer, the handshake signal *mem_addr_rdy* may be asserted low. In the low state, the memory request is stalled until *mem_addr_rdy* is asserted high again.

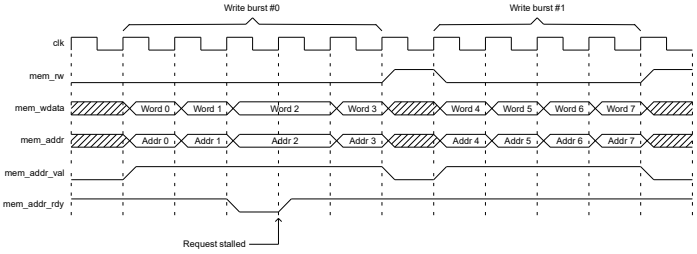

Figure 7: Memory write burst timing (burst size of 4)

The timing is very similar for a read burst. Figure 8 shows a single read burst and corresponding read data returned from memory.

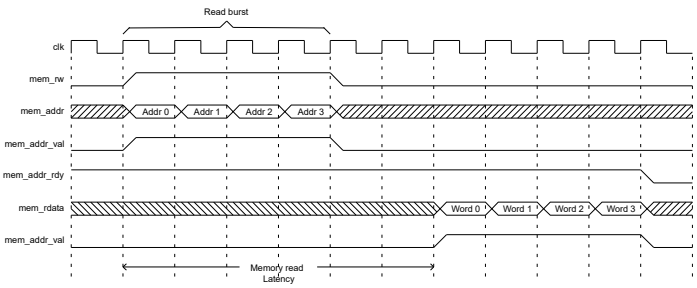

Figure 8: Memory read burst timing (burst size of 4)

5 A larger burst size is advised for synchronous memory types to reduce page-breaks. A burst size of 4 is shown for example only.

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_src_reader.vhd	Reads text-based source video file
mem_model_pack.vhd	Memory model functions
ram_model.vhd	Single port memory model
mem_model_1Mx128bit.vhd	Large 1Mx128 memory model
pipeline_reg.vhd	Pipeline register element
vid_in_reg.vhd	Video input register
vid_out_reg.vhd	Video output register
vid_sync_fifo.vhd	Synchronous pixel FIFO
vid_sync_fifo_reg.vhd	Sync FIFO internal register
ram_dp_w_r.vhd	Dual port RAM component
vid_align_frame.vhd	Aligns pixels to the start of frame
vid_pack128.vhd	Pixel packer
pack_16_to_32.vhd	16-bit to 32-bit packer
pack_24_to_32.vhd	24-bit to 32-bit packer
pack_32_to_32.vhd	32-bit to 32-bit packer
pack_32_to_128.vhd	32-bit to 128-bit packer
vid_frame_fifo.vhd	Main frame-FIFO controller
vid_mem_write.vhd	Memory write burst controller
vid_mem_read.vhd	Memory read burst controller
vid_mem_arb.vhd	Memory R/W arbiter
vid_unpack128.vhd	Pixel unpacker
unpack_32_to_16.vhd	32-bit to 16-bit unpacker
unpack_32_to_24.vhd	32-bit to 24-bit unpacker
unpack_32_to_32.vhd	32-bit to 32-bit unpacker
unpack_128_to_32.vhd	128-bit to 32-bit unpacker
vid_sync_regen.vhd	Video sync generator
vid_uflow_check.vhd	Pixel underflow checker
vid_frame_buffer.vhd	Top-level component
vid_frame_buffer_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 the same order as described in the source file description above.

The VHDL testbench instantiates the VID_FRAME_BUFFER component and the user may modify the generic parameters in order to set up the desired test conditions.

The source video for the simulation is generated by the video source-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_sof,* and *pixin* on a clock-by-clock basis. An example file for a 24-bit/pixel input source might be the following:

```
1  1  000000   # pixel 0, frame 0
1  0  111111   # pixel 1, frame 0
0  0  000000   # don't care!
1  0  222222   # pixel 2, frame 0
1  0  333333   # pixel 3, frame 0
.
.
1  1  000000   # pixel 0 frame 1
1  0  111111   # pixel 1 frame 1 etc.
```

In this example, the first line of the *video_in.txt* file asserts the input signals pixin_val = 1, pixin_sof = 1, and pixin = 0x000000, the second line asserts the input signals pixin_val = 1, pixin_sof = 0, and pixin = 0x111111 etc.

The simulation must be run for at least 30 ms during which time an output text file called *video_out.txt* will be generated. This file contains a sequential list of output pixels in a similar format. Each line defines the state of the signals: *pixout_val*, *pixout_vsync*, *pixout_hsync* and *pixout*. An example output file might be:

```
1  1  1  000000   # pixel 0, frame 0, line 0
1  0  0  111111   # pixel 1, frame 0, line 0
1  0  0  222222   # pixel 2, frame 0, line 0
1  0  0  333333   # pixel 3, frame 0, line 0
1  0  0  444444   # pixel 4, frame 0, line 0
1  0  0  555555   # pixel 5, frame 0, line 0
1  0  0  666666   # pixel 6, frame 0, line 0
1  0  0  777777   # pixel 7, frame 0, line 0
1  0  1  000000   # pixel 0, frame 0, line 1
1  0  0  111111   # pixel 1, frame 0, line 1
.
.
1  1  1  000000   # pixel 0, frame 1, line 0
1  0  0  000000   # pixel 1, frame 1, line 0 etc.
```

In the example test provided, a series of 8 frames of QVGA (320x240) as 24-bit RGB video are buffered in the VFB. Each video frame is numbered 1 to 4 in sequence to ensure that the frame output order is correct. The results of the simulation are shown in Figure 9.

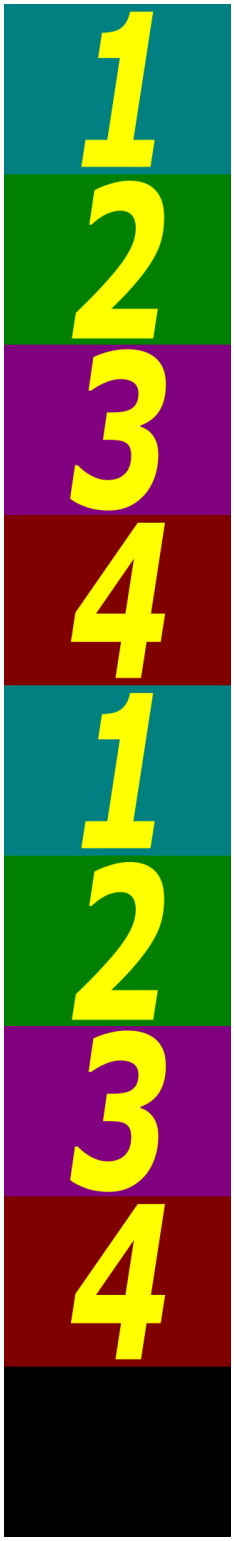

Figure 9: VFB simulation output - 8 frames in sequence