Video Stream Scaler Specifications

Author: David Kronstein
tesla500@hotmail.com

Rev. [1]
February 22, 2011

This page has been intentionally left blank.

Revision History

Rev.	Date	Author	Description
1	February 22 2011	David Kronstein	Document created

Contents

1 Introduction

The Video Stream Scaler (streamScaler) performs resizing of video streams in a low latency manner, resizing with either bilinear or nearest-neighbor modes.

The core offers runtime configuration of input and output resolution, scaling factors, and resize type. Data width, color channels and maximum video resolution can be configured at compile time.

2 Architecture

The heart of the Video Stream Scaler is a Ram FIFO (RFIFO). This operates as a normal FIFO, except instead of data, the input and output are port connections to RAM blocks. A single write port and two read ports are provided, each read port reading *(address) and *(address+1) simultaneously. With sequential video lines written into the RAMs, the four output values are ready for bilinear interpolation.

The RAM fill logic determines which input data to push into the RFIFO and which to discard based on the scaler configuration. All data used by the output section is written to the FIFO, lines which are unused by the output section are discarded.

The Read Control drives the address and read enable for the RAM ports of the RFIFO, as well as advancing the read port of the RFIFO as new output video lines are read out. Blending values are generated that represent how much to interpolate between the four pixels read out of the RFIFO.

The Blend Control takes the blending values and generates four coefficients to be multiplied by the four pixel values from the RFIFO. The sum of these products is the output data. In nearest-neighbor resize mode, only one coefficient is set to 1.0, the rest are set to 0.0, based on the values of the blend values.

See Figure 1 below for a block diagram of the scaler.

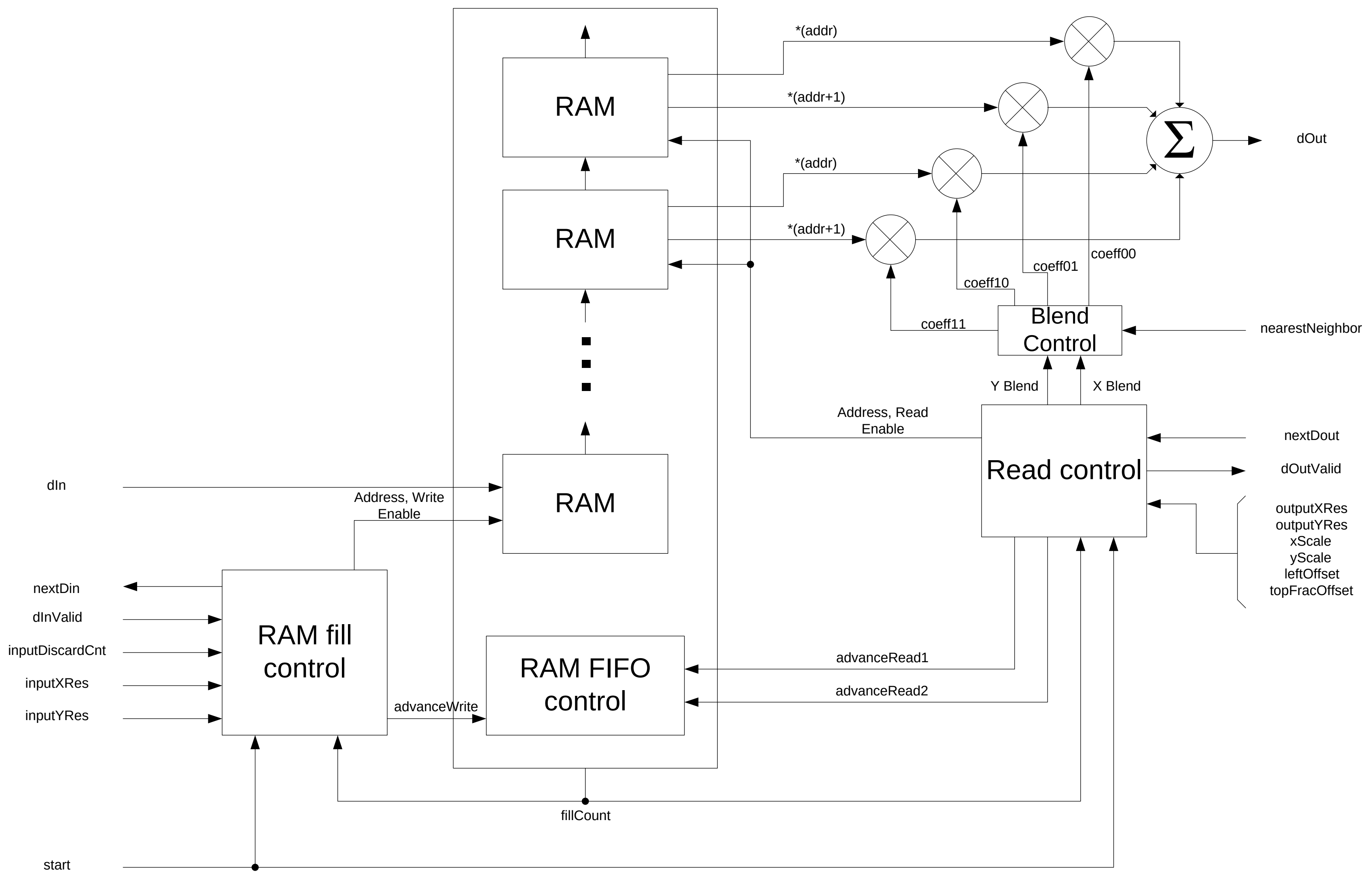

Figure 1 - Video Stream Scaler block diagram

3

Operation

Input and output stream data format

At the start of each video frame, the `start` signal is asserted by the user logic for minimum one clock cycle. This resets the scaler module after the previous frame. The scaler then reads in video data on the input data port. Once sufficient data has been read in, data is presented on the output port. Input data starts at the top left pixel, in a standard raster pattern. Only active video data is put through the scaler. The control signals specify how input data is interpreted.

Control signals must not be changed while the scaler is in operation (reading in or writing out data). Doing so will corrupt the output data for one frame. Control signals should be changed after readout is complete, before the assertion of the start signal for the next frame.

See example waveforms below for information on input and output signals and timing.

C code equivalent of hardware function

The scaling algorithm implemented by streamScaler is described by the following C code:

```
#define       SCALE_FRAC_BITS     14
#define       FRACTION_BITS 8
#define       SCALE_ONE           (1 << SCALE_FRAC_BITS)     //2**SCALE_FRAC_BITS
#define       COEFF_BITS          FRACTION_BITS + 1
#define       COEFF_ONE           (1 << FRACTION_BITS)

uint8 input [INPUT_X_RES*INPUT_Y_RES];
uint8 output [OUTPUT_X_RES*OUTPUT_Y_RES];
uint32 leftOffset;
uint32 topFracOffset
int x, y, xBlend, yBlend;

for(y = 0; y < OUTPUT_Y_RES; y++)
       for(x = 0; x < OUTPUT_X_RES; x++)
       {
              xBlend = (x * xScale + leftOffset) & (SCALE_ONE-1) >> (SCALE_FRAC_BITS - FRACTION_BITS);
              yBlend = (y * yScale + topFracOffset) & (SCALE_ONE-1) >> (SCALE_FRAC_BITS - FRACTION_BITS);

              coeff00 = ((COEFF_ONE - xBlend) * (COEFF_ONE - yBlend)) >> COEFF_BITS;
              coeff01 = (xBlend * (COEFF_ONE - yBlend)) >> COEFF_BITS;
              coeff10 = ((COEFF_ONE - xBlend) * yBlend) >> COEFF_BITS;
              coeff11 = (xBlend * yBlend) >> COEFF_BITS;

              output[x+y*OUTPUT_X_RES] =
                     coeff00 * input[(x*xScale+leftOffset)/SCALE_ONE +
                            (y*yScale+topFracOffset)/SCALE_ONE*INPUT_X_RES] +
                     coeff01 * input[(x*xScale+leftOffset)/SCALE_ONE + 1 +
                            y*yScale+topFracOffset)/SCALE_ONE*INPUT_X_RES] +
                     coeff10 * input[(x*xScale+leftOffset)/SCALE_ONE +
                            ((y*yScale+topFracOffset)/SCALE_ONE + 1)*INPUT_X_RES] +
                     coeff11 * input[(x*xScale+leftOffset)/SCALE_ONE + 1 +
                            ((y*yScale+topFracOffset)/SCALE_ONE + 1)*INPUT_X_RES];
       }
```

Example Waveforms

Overview of data input and output

This example shows scaling up in resolution

```
                    _
start          ____/ \____________________________________________________________________________________________________________________________
               __________ ________ ________ ________ ________ ______________________________________ ________ ________________________ ________ __
dIn            __________X_Line 1_X_Line 2_X_Line 3_X_Line 4_X_________________xx___________________X_Line 5_X____________xx__________X_Line 6_X__
                          ________________________________________________________________________________________________________________________
dInValid       __________/
               ____    ______________________________________                                        ________                          ________
nextDin        _x__\__/                                      \______________________________________/        \________________________/        \__
               ___________________________________________________ ________________ ________________ ________________ ________________ ____________
dOut           ______________________xx___________________________X________L0______X________L1______X________L2______X________L3______X________L4__
                                                                   _______________  _______________  _______________  _______________  ____________
dOutValid      ___________________________________________________/               \/               \/               \/               \/
                                                                  ________________________________________________________________________________
nextdOut       __________________________________________________/
```

User logic sets control signals before start signal. `start` is held high by user logic for 1 or more clock cycles at the beginning of the frame.
The scaler reads in lines until the buffer is full. If specified by control signals, input data is discarded before the buffer begins getting filled. Once two lines have been read in, data can start being read from dOut. Attempting to read early is safe, dOutValid will simply stay low until data is ready.
Continuing to read data from the scaler after the programmed frame size has been read out is safe, and will not cause extra data to be read from the input.

Example of writing data to dIn

```
                ___     ___     ___     ___     ___     ___     ___     ___     ___     ___     ___     ___
clk         ___/   \___/   \___/   \___/   \___/   \___/   \___/   \___/   \___/   \___/   \___/   \___/   \___/
            ____________ _______________ _______ _______ _______________ _______ _______ _______________________
dIn         _____xx_____X______D0_______X__D1___X__D2___X______D3_______X__D4___X__D5___X___________xx__________
                         _______________________________________________________________
dInValid    ____________/                                                               \_______________________
                                 _______________________         _______________________________________________
nextdIn     ____________________/                       \_______/
```

User logic asserts initial data along with dInValid. Scaler acknowledges dIn with nextdIn. User logic must change dIn to the next value when nextdIn is high along with dInValid.

Example of reading data from dOut

```
                   ___     ___     ___     ___     ___     ___     ___     ___     ___     ___     ___     ___     ___     ___     ___     __
clk            ___/   \___/   \___/   \___/   \___/   \___/   \___/   \___/   \___/   \___/   \___/   \___/   \___/   \___/   \___/   \___/
               ____________________________________________ _______ _______ _______________ _______ _______ _______ _______ _______ _______ _
dOut           _____________xx_____________________________X__D0___X__D1___X______D2_______X__D3___X__D4___X__D5___X__D6___X__D7___X__D8___X_
                                                            _______________         _________________________________________________________
dOutValid      ____________________________________________/               \_______/
                            _______________         _________________________________________________________________________________________
nextdOut       ____________/               \_______/
```

Asserting nextDout will request data to be read out from the scaler. If the RFIFO runs out of data, dOutValid will remain low until the buffer becomes full enough. Note the pipeline delay of 4 clock cycles from request signal nextDout to data presented on dOut.

Output line switch

At the end of an output video line, `dOutValid` will go low for one cycle to allow the RFIFO to switch RAMs. If the user logic makes `nextdOut` low after the line is finished being read out (as when reading data out for live display with a horizontal blanking period), this skipped cycle will occur during the time when `nextdOut` is low, and will consequently not be seen by the user logic.

If the last pixel in a line of output video is D9:

```
                 ___     ___     ___     ___     ___     ___     ___     ___
clk          ___/   \___/   \___/   \___/   \___/   \___/   \___/   \___/
             ____ _______ _______ _______ _______ _______ _______ _______ __
dOut         ____X__D7___X__D8___X__D9___X__xx___X__D10__X__D11__X__D12__X__
             ____________________________         __________________________
dOutValid                                \_______/
             _______________________________________________________________
nextdOut
```

Control signals

The operation of the scaler is controlled by the control signals. Refer to section 6 for additional information and a table of all signals.

Minor input Y resolution clipping can be accomplished with `inputDiscardCnt`. Note that large amounts of clipping should be done by changing what data is fed into streamScaler. A number of pixels equal to `inputDiscardCnt` are read in but discarded at the beginning of the image.

The input resolution signals (`inputXRes`, `inputYRes`, `outputXRes`, `outputYRes`) tell the scaler how to interpret and output the pixel stream. These values must be set to the image pixel stream resolution minus 1.

The amount of scaling is controlled by `xScale` and `yScale`. Video is scaled up by `1/xScale` and `1/yScale`. The correct setting for these values is:

If `inputSize` / `outputSize` is an integer:

$2^{SCALE\_FRAC\_BITS} * inputSize / outputSize$

otherwise

$2^{SCALE\_FRAC\_BITS} * (inputSize - 1) / (outputSize - 1) - 1$

`leftOffset` is used to clip the input video in the X direction only, as well as to allow better blending when scaling down by an integer amount.

When clipping, set $leftOffset = offsetInPixels * 2^{SCALE\_FRAC\_BITS}$

If scaling down by an integer amount, set $leftOffset = 2^{(SCALE\_FRAC\_BITS - 1)} - 1$.

If clipping and scaling down by an integer amount, set

$leftOffset = offsetInPixels * 2^{SCALE\_FRAC\_BITS} + 2^{(SCALE\_FRAC\_BITS - 1)} - 1$

If not clipping or scaling down by an integer amount, set `leftOffset = 0`.

`topFracOffset` is used to allow better blending when scaling down by an integer amount. If scaling down by an integer amount, set $topFracOffset = 2^{(SCALE\_FRAC\_BITS - 1)}-1$. If scaling by a non-integer amount, set `topFracOffset = 0`.

`nearestNeighbor` controls scaling method. 0 = bilinear scaling, 1 = nearest neighbor scaling.

Resouce utilization

V1.00 on Altera Cyclone III 3C120:

Configuration	Logic cells	Registers	M9ks	9x9 mult	18x18 mult
10 bits per pixel, 1 color channel, RFIFO size of 3	571	237	9	3	8

Speed

V1.00 on Altera Cyclone III 3C120, speed grade 7, 19% total logic element utilization:

> 108 MHz

4 Registers

This module has no registers.

5

Clocks

Name	Source	Rates (MHz)			Remarks	Description
		Max	Min	Resolution		
`clk`	Input Pad	-	-	-	Module clock must be sufficient to handle the pixel rate of the fastest interface. If scaling up, module clock is determined by output interface. If scaling down, module clock is determined by input interface.	Main module clock, for both input and output data user interfaces.

Table 1: List of clocks

6 IO Ports

User Interface Signals			
Name	I/O	Width	Description
Clock and Reset			
`clk`	I	1	Clock for all module functions
`rst`	I	1	Asynchronous reset
Video data IO			
`dIn`	I	DATA_WIDTH Default: 8	Video input data. Asserted with dInValid. Acknowledged with nextDin while dInValid is high. For multiple color channels (as set by parameter CHANNELS), concatenate all color channels and input into dIn. Eg .dIn({red, green, blue})
`dInValid`	I	1	Asserted by user logic to indicated valid video data is presented at dIn
`nextDin`	O	1	Indicates that streamScaler is ready for data. When nextDin and dInValid are both high, data is clocked in.
`start`	I	1	Resets scaling hardware and flushes all buffers. Apply a 1 or more clock wide pulse at the beginning of each frame. Image data will start being read in after de-asserting `start`. Once RFIFO has sufficient fill, data is available at dOut.

dOut	O	DATA_WIDTH Default: 8	Video output data. Valid data is asserted with dOutValid. Acknowledged by asserting nextDout while dOutvalid is high. For multiple color channels, output data is concatenated in the same order as dIn. Eg .dOut({red, green, blue})
dOutValid	O	1	Asserted by streamScaler to indicate valid output data is presented on dOut.
nextDout	I	1	Indicates that user logic is read for data. When dOutValid and nextDout are both high, output data changes.
Control Interface			
inputDiscardCnt	I	DISCARD_CNT_WIDTH Default: 8	Number of input data to discard before processing data. Used for Y clipping.
inputXRes	I	INPUT_X_RES_WIDTH Default: 11	Horizontal (x) resolution of input video minus 1
inputYRes	I	INPUT_Y_RES_WIDTH Default: 11	Vertical (y) resolution of input video minus 1
outputXRes	I	OUTPUT_X_RES_WIDTH Default: 11	Horizontal (x) resolution of output video minus 1
outputYRes	I	OUTPUT_Y_RES_WIDTH Default: 11	Vertical (y) resolution of output video minus 1
xScale	I	SCALE_INT_BITS + SCALE_FRAC_BITS Default 18	Horizontal scaling factor. Video is scaled by 1/xScale.
yScale	I	SCALE_INT_BITS + SCALE_FRAC_BITS Default 18	Vertical scaling factor. Video is scaled by 1/yScale.
leftOffset	I	OUTPUT_X_RES_WIDTH + SCALE_FRAC_BITS Default: 25	Number of pixels of output image to discard from left side. Format Q OUTPUT_X_RES_WIDTH. SCALE_FRAC_BITS

`topFracOffset`	I	SCALE_FRAC_BITS Default: 14	Fraction of output pixel to discard from top of image. Format Q0. SCALE_FRAC_BITS
`nearestNeighbor`	I	1	Resize type. 0 = bilinear, 1 = nearest neighbor

Table 2: List of IO ports

Appendix A

Parameters

The following are user adjustable parameters for streamScaler:

Name	Default Value	Description
DATA_WIDTH	8	Data width for each channel of video data
CHANNELS	1	Number of color channels
DISCARD_CNT_WIDTH	8	Width of `inputDiscardCnt` signal. Set this to allow the desired amount of data to be discarded.
INPUT_X_RES_WIDTH	11	Width of input horizontal resolution control signal
INPUT_Y_RES_WIDTH	11	Width of input vertical resolution control signal
OUTPUT_X_RES_WIDTH	11	Width of output horizontal resolution control signal
OUTPUT_Y_RES_WIDTH	11	Width of output vertical resolution control signal
FRACTION_BITS	8	Number of bits for fractional component of coefficients. Multiplier with for each coefficient will be 1 plus this value.
SCALE_INT_BITS	4	Width of integer component of scaling factors xScale and yScale. More integer bits will allow scaling down more. Video can be scaled down by a ratio of $1/(2^{\mathrm{SCALE\_INT\_BITS}} - 1/2^{\mathrm{SCALE\_FRAC\_BITS}})$ The width of the input data to the multipliers in the control blocks will be SCALE_INT_BITS + SCALE_FRAC_BITS. For most FPGAs with 18x18 bit multipliers, the sum of these two values should be limited to 18 to allow placement into multiplier blocks.
SCALE_FRAC_BITS	14	Width of fractional component of xScale and yScale. This value controls the accuracy

`BUFFER_SIZE`	4	Number of RAM elements in RFIFO. The following describes the minimum value of BUFFER_SIZE:

Latency requirement	Low latency (live display)	High latency (some output lines may be delayed by one input line period while buffer is filled
Scaling performed		
Scale to same resolution or up only	3	3
Scale down, to same or up	4	3

		If the data source supplying the scaler has high latency, BUFFER_SIZE may need to be increased. If video image exhibits problems, the first thing to try is to increase BUFFER_SIZE

Table 3 - List of user adjustable parameters