Vision HDL Toolbox™

Reference

MATLAB®

R2016b

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

Vision HDL Toolbox™ Reference

Revision History

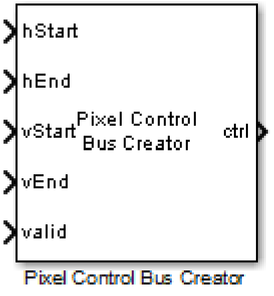
March 2015	Online only	New for Version 1.0 (Release R2015a)
September 2015	Online only	Revised for Version 1.1 (Release R2015b)
March 2016	Online only	Revised for Version 1.2 (Release R2016a)
September 2016	Online only	Revised for Version 1.3 (Release R2016b)

Contents

Blocks — Alphabetical List

Pixel Control Bus Creator

Create control signal bus for use with Vision HDL Toolbox blocks



Library

visionhdlutilities

Description

The Pixel Control Bus Creator block creates a `pixelcontrol` bus. See "Pixel Control Bus".

The block is an implementation of the Simulink® Bus Creator block. See `Bus Creator` for more information.

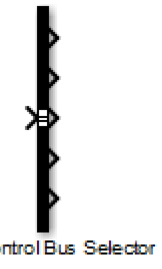
See Also

"Streaming Pixel Interface" | `Frame To Pixels` | `Pixels To Frame`

Introduced in R2015a

Pixel Control Bus Selector

Select signals from control signal bus used by Vision HDL Toolbox blocks



Library

visionhdlutilities

Description

The Pixel Control Bus Selector block selects signals from the `pixelcontrol` bus. See "Pixel Control Bus".

The block is an implementation of the Simulink Bus Selector block. See `Bus Selector` for more information.

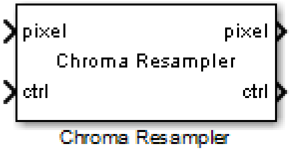
See Also

"Streaming Pixel Interface" | `Frame To Pixels` | `Pixels To Frame`

Introduced in R2015a

Chroma Resampler

Downsample or upsample chrominance component



Library

visionhdlconversions

Description

The Chroma Resampler block downsamples or upsamples a pixel stream.

- Downsampling reduces bandwidth and storage requirements in a video system by combining pixel chrominance components over multiple pixels. You can specify a filter to prevent aliasing, by selecting the default filter or by entering coefficients.
- Upsampling restores a signal to its original rate. You can use interpolation or replication to calculate the extra sample.

This block uses a streaming pixel interface with a bus for synchronization control signals. This interface enables the block to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox™ blocks. The block accepts and returns a scalar pixel value and a bus containing five control signals. These signals indicate the validity of each pixel and the location of each pixel in the frame. To convert a pixel matrix into a pixel stream and these control signals, use the `Frame To Pixels` block. For a full description of the interface, see "Streaming Pixel Interface".

The block accepts luma and chrominance components. The block does not modify the luma component, and applies delay to align it with the resampled chrominance outputs. The rate of the output luma component is the same as the input.

Signal Attributes

Port	Direction	Description	Data Type
pixel	Input/ Output	Single pixel in Y'CbCr color space, specified as a vector of three values. The data type of the output is the same as the data type of the input.	• uint8 or uint16 • fixdt(0,N,0), N = 8,9,...,16 double and single data types are supported for simulation but not for HDL code generation.
ctrl	Input/ Output	Control signals describing the validity of the pixel and the location of the pixel within the frame, specified as a bus containing five signals. See “Pixel Control Bus”.	pixelcontrol

Parameters

Main

Resampling

Resampling operation.

- 4:4:4 to 4:2:2 (default)
- 4:2:2 to 4:4:4

If you select 4:4:4 to 4:2:2, the block performs a downsampling operation. If you select 4:2:2 to 4:4:4, the block performs an upsampling operation.

Antialiasing filter

Lowpass filter to follow a downsample operation.

- Auto (default)
- Property
- None

If you select Auto, the block uses a built-in lowpass filter. If you select Property, the **Horizontal filter coefficients** parameter appears on the dialog box. If you select

`None`, the block does not filter the input signal. This parameter is visible when you set **Resampling** to `4:4:4 to 4:2:2`.

Horizontal filter coefficients

Coefficients for the antialiasing filter.

Enter the coefficients as a vector. The default is `[0.2,0.6,0.2]`. This parameter is visible if you set **Resampling** to `4:4:4 to 4:2:2` and **Antialiasing filter** to `Property`.

Interpolation

Interpolation method for an upsample operation.

- `Linear` (default)
- `Pixel replication`

If you select `Linear`, the block uses linear interpolation to calculate the missing values. If you select `Pixel replication`, the block repeats the chrominance values of the preceding pixel to create the missing pixel. This parameter is visible if you set **Resampling** to `4:2:2 to 4:4:4`.

Data Types

The parameters on this tab appear only when they are relevant. If your selections on the **Main** tab configure the block so that no filter coefficients are needed, or no rounding or overflow is possible, the irrelevant parameter is hidden.

Rounding mode

Rounding method for internal fixed-point calculations. **Rounding mode** appears when you select linear interpolation, or include an antialiasing filter. The default is `Floor`.

Overflow mode

Overflow action for internal fixed-point calculations. Overflow can occur when you include an antialiasing filter. The default is `Wrap`.

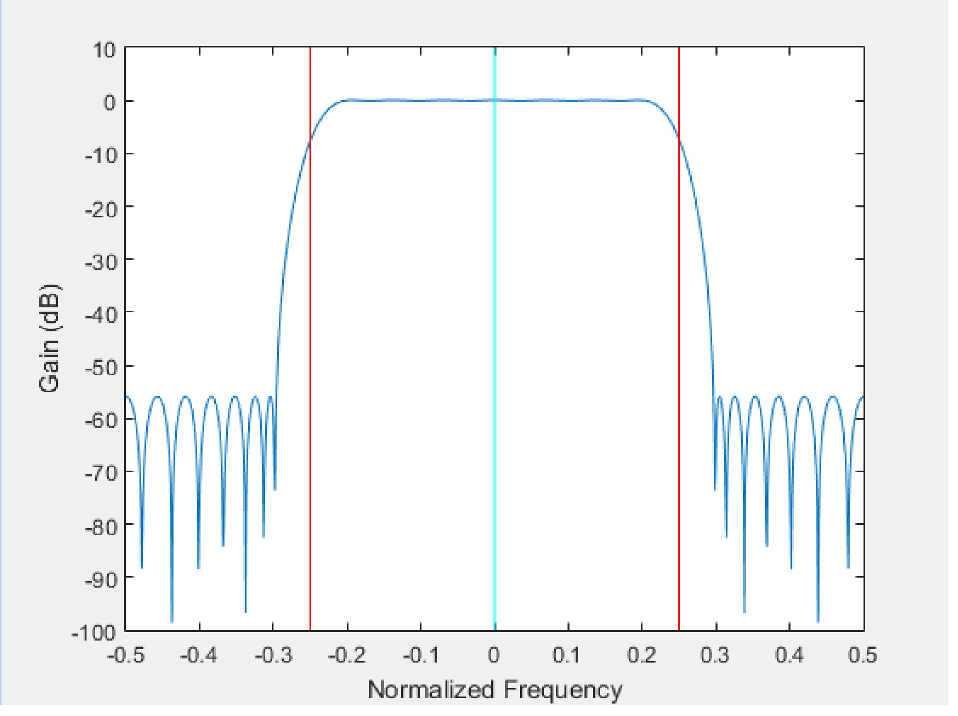
Filter coefficients

Data type for the antialiasing filter coefficients.

The default is `fixdt(1,16,0)`. This parameter is visible when you set **Antialiasing filter** to `Auto` or `Property`.

Algorithm

The default antialiasing filter is a 29-tap lowpass filter that matches the default Chroma Resampling block in Computer Vision System Toolbox™. In the frequency response of this filter, the passband, [-0.25 0.25], occupies half of the total bandwidth. This filter suppresses aliasing after `4:4:4 to 4:2:2` downsampling.



Whether you use the default filter or specify your own coefficients, the filter is implemented in HDL using a fully parallel architecture. HDL code generation takes advantage of symmetric, unity, or zero-value coefficients to reduce the number of multipliers.

The block pads the edge of the image with symmetric pixel values. See “Edge Padding”. Also, if the frame is an odd number of pixels wide, the block symmetrically pads the line. This accommodation makes the block more resilient to video timing variation.

Latency

The latency is the number of cycles between the first valid input pixel and the first valid output pixel. When you use an antialiasing filter, the latency depends on the size and value of the filter coefficients. The FIR delay can be less than the number of coefficients because the block optimizes out duplicate or zero-value coefficients.

Block Configuration	Latency
Downsample (4:4:4 to 4:2:2), no filter	3
Downsample (4:4:4 to 4:2:2), with filter	4+(N/2)+FIR delay, N = number of filter coefficients
Upsample (4:2:2 to 4:4:4), replication	3
Upsample (4:2:2 to 4:4:4), interpolation	5

For example, the latency for a downsample using the default filter is 30 cycles.

Note: The horizontal blanking interval of the pixel stream format must be greater than the latency of the block. This interval enables the block to finish processing one line before it starts processing the next one. If you are using a custom video format, set the horizontal blanking interval using the `Frame To Pixels` block parameters. The horizontal blanking interval is equal to **Total pixels per line** – **Active pixels per line**, or, equivalently, **Front porch** + **Back porch**. Standard streaming video formats use a horizontal blanking interval of about 25% of the frame width. This interval is much larger than the filters applied to each frame.

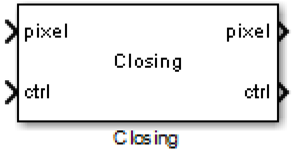
See Also

visionhdl.ChromaResampler | `Chroma Resampling` | `Frame To Pixels`

Introduced in R2015a

Closing

Morphological closing of binary pixel data



Library

visionhdlmorph

Description

The Closing block performs morphological dilation, followed by morphological erosion, using the same neighborhood for both calculations. The block operates on a stream of binary intensity values.

This block uses a streaming pixel interface with a bus for synchronization control signals. This interface enables the block to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox blocks. The block accepts and returns a scalar pixel value and a bus containing five control signals. These signals indicate the validity of each pixel and the location of each pixel in the frame. To convert a pixel matrix into a pixel stream and these control signals, use the `Frame To Pixels` block. For a full description of the interface, see “Streaming Pixel Interface”.

Signal Attributes

Port	Direction	Description	Data Type
`pixel`	Input/ Output	Single image pixel, specified as a scalar binary value.	`boolean`
`ctrl`	Input/ Output	Control signals describing the validity of the pixel and the	`pixelcontrol`

Port	Direction	Description	Data Type
		location of the pixel within the frame, specified as a bus containing five signals. See “Pixel Control Bus”.	

Parameters

Neighborhood

Pixel neighborhood, specified as a matrix or vector of ones and zeros.

The block supports neighborhoods of up to 32×32 pixels. To use a structuring element, specify **Neighborhood** as `getnhood(strel(`*`shape`*`))`.

The default is `[0,1,0;1,1,1;0,1,0]`.

Line buffer size

Size of the line memory buffer, specified as a scalar integer.

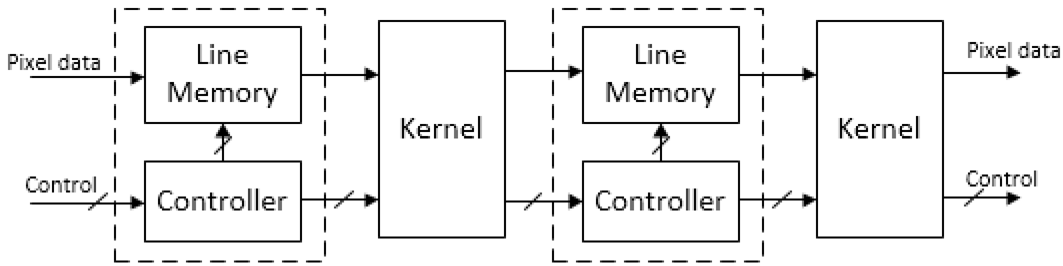
Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the block uses the next largest power of two. The block allocates (*neighborhood lines* – 1)-by-**Line buffer size** memory locations to store the pixels. The default is 2048.

Algorithm

The block pads the image with zeroes for the dilation operation, and with ones for the erosion operation. See “Edge Padding”.

Latency

The total latency of the block is the line buffer latency plus the latency of the kernel calculation. Morphological closing is a compound operation. Therefore, this block contains a second line buffer between the dilation kernel and the erosion kernel. To determine the exact latency for any configuration of the block, monitor the number of time steps between the input control signals and the output control signals.



The latency of the line memory includes edge padding. The latency of the kernel depends on the neighborhood size.

Note: The horizontal blanking interval of the pixel stream format must be greater than the latency of the block. This interval enables the block to finish processing one line before it starts processing the next one. If you are using a custom video format, set the horizontal blanking interval using the `Frame To Pixels` block parameters. The horizontal blanking interval is equal to **Total pixels per line** – **Active pixels per line**, or, equivalently, **Front porch** + **Back porch**. Standard streaming video formats use a horizontal blanking interval of about 25% of the frame width. This interval is much larger than the filters applied to each frame.

See Also

visionhdl.Closing | `Closing` | `Dilation` | `Erosion` | `Frame To Pixels` | `Opening`

More About

- "Morphological Dilation and Erosion"
- "Structuring Elements"

Introduced in R2015a

Color Space Converter

Convert color information between color spaces

Library

visionhdlconversions

Description

The Color Space Converter block converts between R'G'B' and Y'CbCr color spaces, and also converts R'G'B' to intensity.

This block uses a streaming pixel interface with a bus for synchronization control signals. This interface enables the block to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox blocks. The block accepts and returns a scalar pixel value and a bus containing five control signals. These signals indicate the validity of each pixel and the location of each pixel in the frame. To convert a pixel matrix into a pixel stream and these control signals, use the `Frame To Pixels` block. For a full description of the interface, see "Streaming Pixel Interface".

Note: The Color Space Converter block operates on gamma-corrected color spaces. However, to simplify use of the block, the block and mask labels do not include the prime notation.

Signal Attributes

Port	Direction	Description	Data Type
`pixel`	Input/ Output	Single image pixel, specified by a vector of three values representing	• `uint8` or `uint16`

Port	Direction	Description	Data Type
		R'G'B' or Y'CbCr, or a scalar value representing intensity. The data type of the output is the same as the data type of the input.	• `fixdt(0,N,0)`, $N = 8,9,...,16$ `double` and `single` data types are supported for simulation but not for HDL code generation.
`ctrl`	Input/ Output	Control signals describing the validity of the pixel and the location of the pixel within the frame, specified as a bus containing five signals. See “Pixel Control Bus”.	`pixelcontrol`

Parameters

Conversion

Conversion that the block performs on the input video stream.

- `RGB to YCbCr` (default)
- `YCbCr to RGB`
- `RGB to intensity`

The block accepts input as a vector of three values representing a single pixel. If you choose `RGB to intensity`, the output is a scalar value. Otherwise, the output is a vector of three values.

Use conversion specified by

Conversion equation to use on the input video stream. This parameter does not apply when you set **Conversion** to `RGB to intensity`.

- `Rec. 601 (SDTV)` (default)
- `Rec. 709 (HDTV)`

Scanning standard

Scanning standard to use for HDTV conversion. This parameter applies when you set **Use conversion specified by** to `Rec. 709 (HDTV)`.

- `1250/50/2:1` (default)
- `1125/60/2:1`

Algorithm

Conversion Between R'G'B' and Y'CbCr Color Spaces

The following equations define R'G'B' to Y'CbCr conversion and Y'CbCr to R'G'B' conversion:

$$\begin{bmatrix} Y' \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \mathrm{A} \times \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}$$

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \mathrm{B} \times \left(\begin{bmatrix} Y' \\ Cb \\ Cr \end{bmatrix} - \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} \right)$$

The values in matrices A and B are based on your choices for the **Use conversion specified by** and **Scanning standard** parameters.

Matrix	Use conversion specified by = Rec. 601 (SDTV)	Use conversion specified by = Rec. 709 (HDTV)	
		Scanning standard = 1125/60/2:1	Scanning standard = 1250/50/2:1
A	$\begin{bmatrix} 0.25678824 & 0.50412941 & 0.09790588 \\ -0.1482229 & -0.29099279 & 0.43921569 \\ 0.43921569 & -0.36778831 & -0.07142737 \end{bmatrix}$	$\begin{bmatrix} 0.18258588 & 0.61423059 & 0.06200706 \\ -0.10064373 & -0.33857195 & 0.43921569 \\ 0.43921569 & -0.39894216 & -0.04027352 \end{bmatrix}$	$\begin{bmatrix} 0.25678824 & 0.50412941 & 0.09790588 \\ -0.1482229 & -0.29099279 & 0.43921569 \\ 0.43921569 & -0.36778831 & -0.07142737 \end{bmatrix}$
B	$\begin{bmatrix} 1.1643836 & 0 & 1.5960268 \\ 1.1643836 & -0.39176229 & -0.81296765 \\ 1.1643836 & 2.0172321 & 0 \end{bmatrix}$	$\begin{bmatrix} 1.16438356 & 0 & 1.79274107 \\ 1.16438356 & -0.21324861 & -0.53290933 \\ 1.16438356 & 2.11240179 & 0 \end{bmatrix}$	$\begin{bmatrix} 1.1643836 & 0 & 1.5960268 \\ 1.1643836 & -0.39176229 & -0.81296765 \\ 1.1643836 & 2.0172321 & 0 \end{bmatrix}$

Conversion from R'G'B' to Intensity

The following equation defines conversion from R'G'B' color space to intensity:

$$\text{intensity} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}$$

Data Types

For fixed-point and integer input, the block converts matrix *A* to `fixdt(1,17,16)`, and matrix *B* to `fixdt(1,17,14)`.

For double or single input, the block applies the conversion matrices in double type, and scales the Y'CbCr offset vector (`[16,128,128]`) by 1/255. The block saturates double or single R'G'B' and intensity outputs to the range `[0,1]`.

The Y'CbCr standard includes headroom and footroom. For 8-bit data, luminance values 16–235, and chrominance values 16–240, are valid. The Color Space Converter block pins out-of-range input to these limits before calculating the conversion. The block scales the offset vector and the allowed headroom and footroom depending on the word length of the input signals. For example, when you convert a Y'CbCr input of type `fixdt(0,10,0)` to R'G'B', the block multiplies the offset vector by $2^{(10-8)} = 4$. As a result, the valid luminance range becomes 64–940 and the valid chrominance range becomes 64–960.

Latency

Blocks with R'G'B' input have a latency of 9 cycles. Blocks with Y'CbCr input have a latency of 10 cycles because one cycle is required to check for and correct headroom and footroom violations.

Note: The horizontal blanking interval of the pixel stream format must be greater than the latency of the block. This interval enables the block to finish processing one line before it starts processing the next one. If you are using a custom video format, set the horizontal blanking interval using the `Frame To Pixels` block parameters. The horizontal blanking interval is equal to **Total pixels per line** – **Active pixels per line**, or, equivalently, **Front porch** + **Back porch**. Standard streaming video formats use a horizontal blanking interval of about 25% of the frame width. This interval is much larger than the filters applied to each frame.

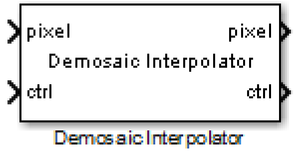
See Also

visionhdl.ColorspaceConverter | `Color Space Conversion` | `Frame To Pixels`

Introduced in R2015a

Demosaic Interpolator

Construct RGB pixel data from Bayer pattern pixels



Library

visionhdlconversions

Description

The Demosaic Interpolator block provides a Bayer pattern interpolation filter for streaming video data. The block implements the calculations using hardware-efficient, multiplier-free algorithms for HDL code generation. You can select a low complexity bilinear interpolation, or a moderate complexity gradient-corrected bilinear interpolation.

- When you choose bilinear interpolation, the block operates on a 3×3 pixel window using only additions and bit shifts.
- When you choose gradient correction, the block operates on a 5×5 pixel window. The calculation is performed using bit shift, addition, and low order Canonical Signed Digit (CSD) multiplication.

This block uses a streaming pixel interface with a bus for synchronization control signals. This interface enables the block to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox blocks. The block accepts and returns a scalar pixel value and a bus containing five control signals. These signals indicate the validity of each pixel and the location of each pixel in the frame. To convert a pixel matrix into a pixel stream and these control signals, use the `Frame To Pixels` block. For a full description of the interface, see "Streaming Pixel Interface".

Signal Attributes

Port	Direction	Description	Data Type
pixel	Input	Single pixel, specified as a scalar value.	• uint or int • fixdt(0,N,0) double and single data types are supported for simulation but not for HDL code generation.
ctrl	Input/ Output	Control signals describing the validity of the pixel and the location of the pixel within the frame, specified as a bus containing five signals. See "Pixel Control Bus".	pixelcontrol
pixel	Output	Single pixel in RGB color space, returned as a vector of three values.	Same as the inputpixel

Parameters

Interpolation algorithm

Algorithm the block uses to calculate the missing pixel values.

- Bilinear — Average of the pixel values in the surrounding 3×3 neighborhood.
- Gradient-corrected linear (default) — Bilinear average, corrected for intensity gradient.

Sensor alignment

Color sequence of the pixels in the input stream.

Select the sequence of R, G and B pixels that correspond to the 2-by-2 block of pixels in the top-left corner of the input image. Specify the sequence in left-to-right, top-to-bottom order. For instance, the default RGGB represents an image with this pattern.

R	G
G	B

Line buffer size

Size of the line memory buffer, specified as a scalar integer.

Choose a power of 2 that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the object uses the next largest power of two. When you select `Bilinear` interpolation, the block allocates 2-by-**Line buffer size** memory locations. When you select `Gradient-corrected linear` interpolation, the block allocates 4-by-**Line buffer size** memory locations. The default value is 2048.

Algorithm

The block pads the edges of the image with symmetric pixel values. See “Edge Padding”.

Interpolation

Bilinear Interpolation

The block interpolates the missing color values using a 3×3 neighborhood. The average is calculated over the adjacent two pixels or four pixels, depending on the sensor color pattern. The block implements this algorithm using only add and shift operations.

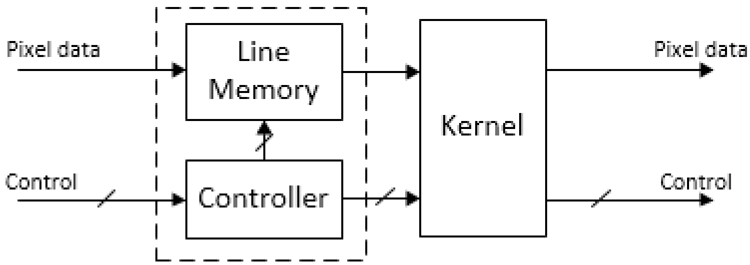
Gradient-Corrected Linear Interpolation

Gradient correction improves interpolation performance across edges by taking advantage of the correlation between the color channels. The block calculates the missing color values using bilinear interpolation, and then modifies the value corresponding to the intensity gradient calculated over a 5×5 neighborhood. The block applies the gradient correction using a fixed set of filter kernels. The filter coefficients were designed empirically to perform well over a wide range of image data. The coefficients are multiples of powers of two to enable an efficient hardware implementation. See [1].

Latency

The block buffers one line of input pixels before starting bilinear interpolation calculations. The gradient correction calculation starts after the block buffers 2 lines.

The latency of the block is the line buffer latency plus the latency of the kernel calculation. The line buffer latency includes edge padding. To determine the exact latency for any configuration of the block, you can measure the number of time steps between the input control signals and the output control signals .



Note: The horizontal blanking interval of the pixel stream format must be greater than the latency of the block. This interval enables the block to finish processing one line before it starts processing the next one. If you are using a custom video format, set the horizontal blanking interval using the `Frame To Pixels` block parameters. The horizontal blanking interval is equal to **Total pixels per line** – **Active pixels per line**, or, equivalently, **Front porch** + **Back porch**. Standard streaming video formats use a horizontal blanking interval of about 25% of the frame width. This interval is much larger than the filters applied to each frame.

References

[1] Malvar, Henrique S., Li-wei He, and Ross Cutler. “High-Quality Linear Interpolation for Demosaicing of Bayer-Patterned Color Images.” *Microsoft Research*, May 2004. http://research.microsoft.com/pubs/102068/Demosaicing_ICASSP04.pdf.

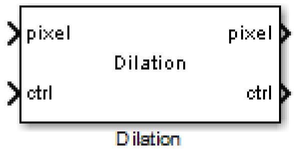
See Also

visionhdl.DemosaicInterpolator | `Demosaic` | `Frame To Pixels`

Introduced in R2015a

Dilation

Morphological dilation of binary pixel data



Library

visionhdlmorph

Description

The Dilation block replaces each pixel with the local maximum of the neighborhood around the pixel. The block operates on a stream of binary intensity values.

This block uses a streaming pixel interface with a bus for synchronization control signals. This interface enables the block to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox blocks. The block accepts and returns a scalar pixel value and a bus containing five control signals. These signals indicate the validity of each pixel and the location of each pixel in the frame. To convert a pixel matrix into a pixel stream and these control signals, use the `Frame To Pixels` block. For a full description of the interface, see "Streaming Pixel Interface".

Signal Attributes

Port	Direction	Description	Data Type
`pixel`	Input/ Output	Single image pixel, specified as a scalar binary value.	`boolean`
`ctrl`	Input/ Output	Control signals describing the validity of the pixel and the	`pixelcontrol`

Port	Direction	Description	Data Type
		location of the pixel within the frame, specified as a bus containing five signals. See "Pixel Control Bus".	

Parameters

Neighborhood

Pixel neighborhood, specified as a matrix or vector of ones and zeros.

The block supports neighborhoods of up to 32×32 pixels. To use a structuring element, specify **Neighborhood** as `getnhood(strel(shape))`.

The default is `ones(3,3)`.

Line buffer size

Size of the line memory buffer, specified as a scalar integer.

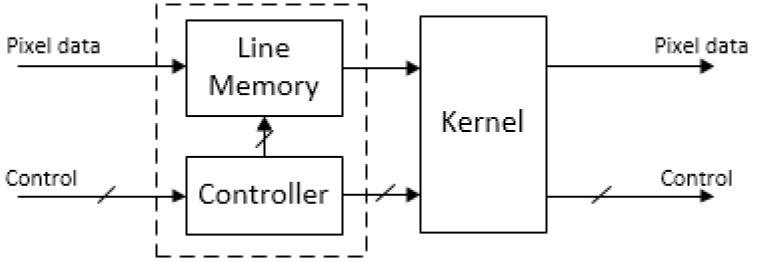
Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the block uses the next largest power of two. The block allocates (*neighborhood lines* – 1)-by-**Line buffer size** memory locations to store the pixels. The default is 2048.

Algorithm

The block pads the image with zeroes for the dilation operation. See "Edge Padding".

Latency

The latency of the block is the line buffer latency plus the latency of the kernel calculation. The line buffer latency includes edge padding. To determine the exact latency for any configuration of the block, you can measure the number of time steps between the input control signals and the output control signals .



The latency of the kernel depends on the neighborhood size.

Note: The horizontal blanking interval of the pixel stream format must be greater than the latency of the block. This interval enables the block to finish processing one line before it starts processing the next one. If you are using a custom video format, set the horizontal blanking interval using the `Frame To Pixels` block parameters. The horizontal blanking interval is equal to **Total pixels per line** – **Active pixels per line**, or, equivalently, **Front porch** + **Back porch**. Standard streaming video formats use a horizontal blanking interval of about 25% of the frame width. This interval is much larger than the filters applied to each frame.

See Also

visionhdl.Dilation | `Erosion` | `Frame To Pixels`

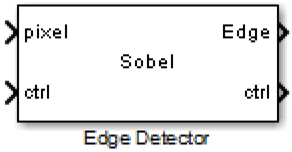
More About

- "Morphological Dilation and Erosion"
- "Structuring Elements"

Introduced in R2015a

Edge Detector

Find edges of objects



Library

visionhdlanalysis

Description

The Edge Detector block finds the edges in a grayscale pixel stream using the Sobel, Prewitt, or Roberts method. The block convolves the input pixels with derivative approximation matrices to find the gradient of pixel magnitude along two orthogonal directions. It then compares the sum of the squares of the gradients to the square of a configurable threshold to determine if the gradients represent an edge.

By default, the block returns a binary image as a stream of pixel values. A pixel value of 1 indicates that the pixel is an edge. You can disable the edge output. You can also enable output of the gradient values in the two orthogonal directions at each pixel.

This block uses a streaming pixel interface with a bus for synchronization control signals. This interface enables the block to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox blocks. The block accepts and returns a scalar pixel value and a bus containing five control signals. These signals indicate the validity of each pixel and the location of each pixel in the frame. To convert a pixel matrix into a pixel stream and these control signals, use the `Frame To Pixels` block. For a full description of the interface, see “Streaming Pixel Interface”.

Signal Attributes

Port	Direction	Description	Data Type
pixel	Input	Single image pixel, specified as a scalar value.	• uint or int • fixdt() double and single data types are supported for simulation but not for HDL code generation.
ctrl	Input	Control signals describing the validity of the pixel and the location of the pixel within the frame, specified as a bus containing five signals. See “Pixel Control Bus”.	pixelcontrol
Th	Input (optional)	Threshold value that defines an edge, specified as a scalar. The block compares the square of this value to the sum of the squares of the gradients.	• uint or int • fixdt() double and single data types are supported for simulation but not for HDL code generation.
Edge	Output (optional)	Pixel value indicating whether the pixel is an edge, returned as a scalar binary value.	boolean
Gv, Gh	Output (optional)	Vertical and horizontal gradient values. These ports are visible when you choose the Sobel or Prewitt method.	• uint or int • fixdt() double and single data types are supported for simulation but not for HDL code generation.
G45, G135	Output (optional)	Orthogonal gradient values. These ports are visible when you choose the Roberts method.	Same as Gv, Gh
ctrl	Output	Control signals describing the validity of the pixel and the location of the pixel within	pixelcontrol

Port	Direction	Description	Data Type
		the frame, specified as a bus containing five signals. See “Pixel Control Bus”.	

Parameters

Main

Method

Edge detection algorithm.

Select `Sobel`, `Prewitt`, or `Roberts` method.

Output the binary image

Enable the `Edge` output port.

When selected, the block returns a stream of binary pixels representing the edges detected in the input frame. By default, this check box is selected. You must select at least one of **Output the binary image** and **Output the gradient components**.

Output the gradient components

Enable the gradient output ports.

When selected, two output ports return values representing the gradients calculated in the two orthogonal directions. By default, this check box is not selected. When you set **Method** to `Sobel` or `Prewitt`, the output ports `Gv` and `Gh` appear on the block. When you set **Method** to `Roberts`, the output ports `G45` and `G135` appear on the block.

You must select at least one of **Output the binary image** and **Output the gradient components**.

Source of threshold value

Source for the gradient threshold value that indicates an edge.

You can set the threshold from an input port or from the dialog box. The default value is `Property`. If you select `Input port`, the `Th` port appears on the block icon.

Threshold value

Gradient threshold value that indicates an edge.

The block compares the square of this value to the sum of the squares of the gradients. The block casts this value to the data type of the gradients. The default value is 20. This option is visible when you set **Source of threshold value** to `Property`.

Line buffer size

Size of the line memory buffer, specified as a scalar integer.

Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the block uses the next largest power of two. The block allocates (*N* - *1*)-by-**Line buffer size** memory locations to store the pixels, where *N* is the number of lines in the differential approximation matrix. If you set **Method** to `Sobel` or `Prewitt`, then *N* is 3. If you set **Method** to `Roberts`, then *N* is 2. The default value is 20.

Data Types

Rounding mode

Rounding mode for internal fixed-point calculations. The default is `Floor`.

Overflow mode

Overflow mode for internal fixed-point calculations. The default is `Wrap`.

Gradient Data Type

Data type for the two gradient output ports.

If you select the **Output the gradient components** check box on the **Main** tab, the **Gradient Data Type** appears on this tab. The default is **Inherit via internal rule**, which means the block automatically chooses full-precision data types.

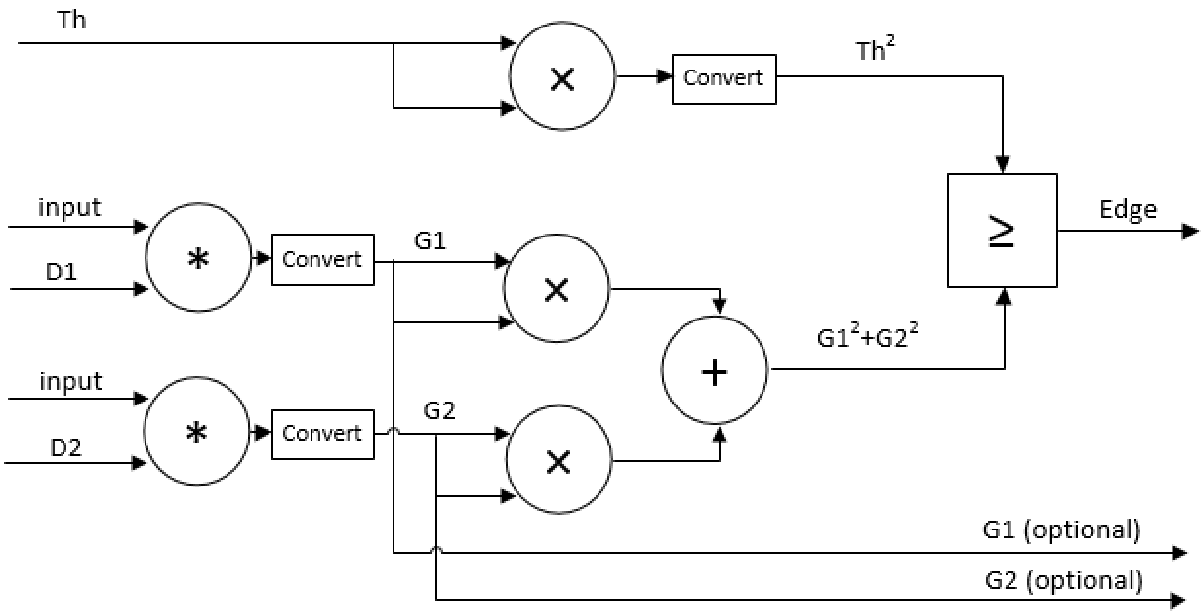
Algorithm

The Edge Detector block provides three methods for detecting edges in an input image. The methods use different derivative approximation matrices to find two orthogonal gradients. The Sobel and Prewitt methods calculate the gradient in horizontal and vertical directions. The Roberts method calculates the gradients at 45 degrees and 135 degrees. The block uses the same matrices as the Edge Detection block in Computer Vision System Toolbox.

Method	Direction 1	Direction 2
Sobel	$\frac{1}{8}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$	$\frac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$
Prewitt	$\frac{1}{6}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$	$\frac{1}{6}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$
Roberts	$\frac{1}{2}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$\frac{1}{2}\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$

Note: The Prewitt coefficients require extra bits of precision because they are not powers of two. The block uses 16 bits to represent the Prewitt coefficients. For 8-bit input, the default size of the full-precision gradients is 27 bits. When using the Prewitt method, a good practice is to reduce the word length used for the gradient calculation. Select the **Output the gradient components** check box, and then on the **Data Types** tab, specify a smaller word length using **Gradient Data Type**.

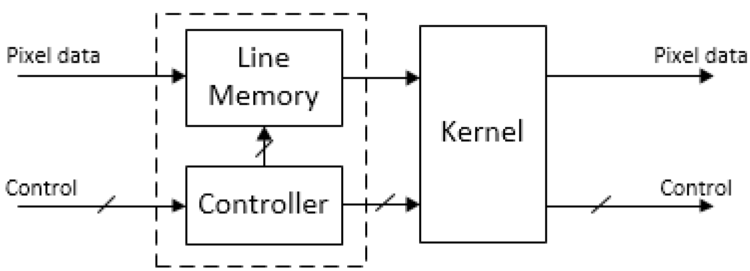
The block convolves the neighborhood of the input pixel with the derivative matrices, D1 and D2. It then compares the sum of the squares of the gradients to the square of the threshold. Computing the square of the threshold avoids constructing a square root circuit. The block casts the gradients to the type you specified on the **Data Types** tab. The type conversion on the square of the threshold matches the type of the sum of the squares of the gradients.



The block pads the edge of the image with symmetric pixel values. See “Edge Padding”.

Latency

The latency of the block is the line buffer latency plus the latency of the kernel calculation. The line buffer latency includes edge padding. To determine the exact latency for any configuration of the block, you can measure the number of time steps between the input control signals and the output control signals .



Note: The horizontal blanking interval of the pixel stream format must be greater than the latency of the block. This interval enables the block to finish processing one line before it starts processing the next one. If you are using a custom video format, set the horizontal blanking interval using the `Frame To Pixels` block parameters. The horizontal blanking interval is equal to **Total pixels per line** – **Active pixels per line**, or, equivalently, **Front porch** + **Back porch**. Standard streaming video formats use a horizontal blanking interval of about 25% of the frame width. This interval is much larger than the filters applied to each frame.

See Also

visionhdl.EdgeDetector | `Edge Detection` | `Frame To Pixels`

Related Examples

- “Edge Detection and Image Overlay”

Introduced in R2015a

Erosion

Morphological erosion of binary pixel data

Library

visionhdlmorph

Description

The Erosion block replaces each pixel with the local minimum of the neighborhood around the pixel. The block operates on a stream of binary intensity values.

This block uses a streaming pixel interface with a bus for synchronization control signals. This interface enables the block to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox blocks. The block accepts and returns a scalar pixel value and a bus containing five control signals. These signals indicate the validity of each pixel and the location of each pixel in the frame. To convert a pixel matrix into a pixel stream and these control signals, use the `Frame To Pixels` block. For a full description of the interface, see “Streaming Pixel Interface”.

Signal Attributes

Port	Direction	Description	Data Type
`pixel`	Input/ Output	Single image pixel, specified as a scalar binary value.	`boolean`
`ctrl`	Input/ Output	Control signals describing the validity of the pixel and the	`pixelcontrol`

Port	Direction	Description	Data Type
		location of the pixel within the frame, specified as a bus containing five signals. See “Pixel Control Bus”.	

Parameters

Neighborhood

Pixel neighborhood, specified as a matrix or vector of ones and zeros.

The block supports neighborhoods of up to 32×32 pixels. To use a structuring element, specify **Neighborhood** as `getnhood(strel(`*`shape`*`))`.

The default is `ones(3,3)`.

Line buffer size

Size of the line memory buffer, specified as a scalar integer.

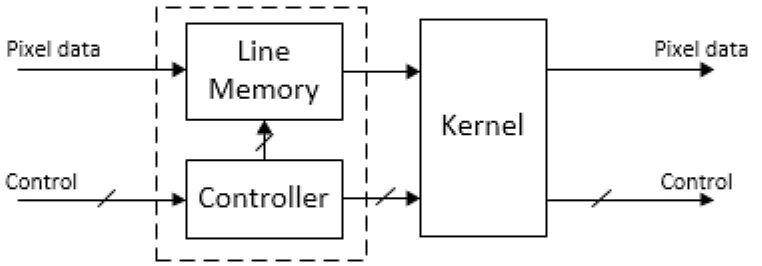
Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the block uses the next largest power of two. The block allocates (*neighborhood lines* – 1)-by-**Line buffer size** memory locations to store the pixels. The default is 2048.

Algorithm

The block pads the edge of the image with ones for the erosion operation. See “Edge Padding”.

Latency

The latency of the block is the line buffer latency plus the latency of the kernel calculation. The line buffer latency includes edge padding. To determine the exact latency for any configuration of the block, you can measure the number of time steps between the input control signals and the output control signals .



The latency of the kernel depends on the neighborhood size.

Note: The horizontal blanking interval of the pixel stream format must be greater than the latency of the block. This interval enables the block to finish processing one line before it starts processing the next one. If you are using a custom video format, set the horizontal blanking interval using the `Frame To Pixels` block parameters. The horizontal blanking interval is equal to **Total pixels per line** – **Active pixels per line**, or, equivalently, **Front porch** + **Back porch**. Standard streaming video formats use a horizontal blanking interval of about 25% of the frame width. This interval is much larger than the filters applied to each frame.

See Also

visionhdl.Erosion | `Dilation` | `Erosion` | `Frame To Pixels`

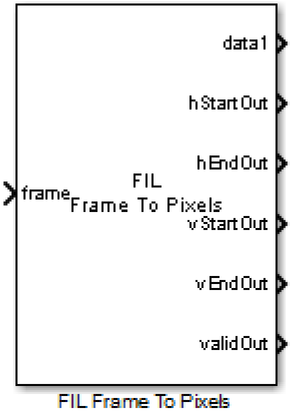
More About

- "Morphological Dilation and Erosion"
- "Structuring Elements"

Introduced in R2015a

FIL Frame To Pixels

Convert frame-based video to pixel stream for FPGA-in-the-loop



Library

visionhdlio

Description

The FIL Frame To Pixels block performs the same frame-to-pixel conversion as the `Frame To Pixels` block. In addition, you can configure the width of the output vector to be a single pixel, a line, or an entire frame. The block returns control signals in vectors of the same width as the pixel data. This optimization makes more efficient use of the communication link between the FPGA board and your Simulink simulation when using FPGA-in-the-loop (FIL). To run FPGA-in-the-loop, you must have an HDL Verifier™ license.

When you generate a programming file for a FIL target in Simulink, the tool creates a model to compare the FIL simulation with your Simulink design. For Vision HDL Toolbox designs, the FIL block in that model replicates the pixel-streaming interface to send one pixel at a time to the FPGA. You can modify the autogenerated model to use the FIL Frame To Pixels and FIL Pixels To Frame blocks to improve communication bandwidth with the FPGA board by sending one frame at a time. For how to modify the autogenerated model, see "FPGA-in-the-Loop".

Specify the same video format and vector size for the FIL Frames To Pixels block and the FIL Pixels To Frame block.

Signal Attributes

Port	Direction	Description	Data Type
`matrix`	Input	Full image, specified as an **Active pixels per line**-by-**Active video lines**-by-*N* matrix. The height and width are the dimensions of the active image specified in **Video format**. *N* is the **Number of components** used to express a single pixel.	• `uint` or `int` • `fixdt()` • `boolean` • `double` or `single`
`data1,...,da`	Output	Image pixels, returned as a vector of *M* values, where *M* is the width of the **Output vector format**. There are *N* data ports, where *N* is the **Number of components**.	Specified by **Data type**
`hStartOut`	Output	Control signal indicating whether each pixel is the first pixel in a horizontal line of a frame, returned as a vector of *M* values.	`boolean`
`hEndOut`	Output	Control signal indicating whether each pixel is the last pixel in a horizontal line of a frame, returned as a vector of *M* values.	`boolean`
`vStartOut`	Output	Control signal indicating whether each pixel is the first pixel in the first (top) line of a frame, returned as a vector of *M* values.	`boolean`
`vEndOut`	Output	Control signal indicating whether each pixel is the last pixel in the last (bottom) line of a frame, returned as a vector of *M* values.	`boolean`

Port	Direction	Description	Data Type
validOut	Output	Control signal indicating the validity of the output pixel, returned as a vector of M values.	boolean

Parameters

Number of components

Component values of each pixel. The pixel can be represented by 1, 3, or 4 components. Set to 1 for grayscale video. Set to 3 for color video, for example, {R,G,B} or {Y,Cb,Cr}. Set to 4 to use color with an alpha channel for transparency. The output is an **Active pixels per line**-by-**Active video lines**-by-**Number of components** image matrix.

Data type

Output pixel data type. The default is `uint8`.

Output vector format

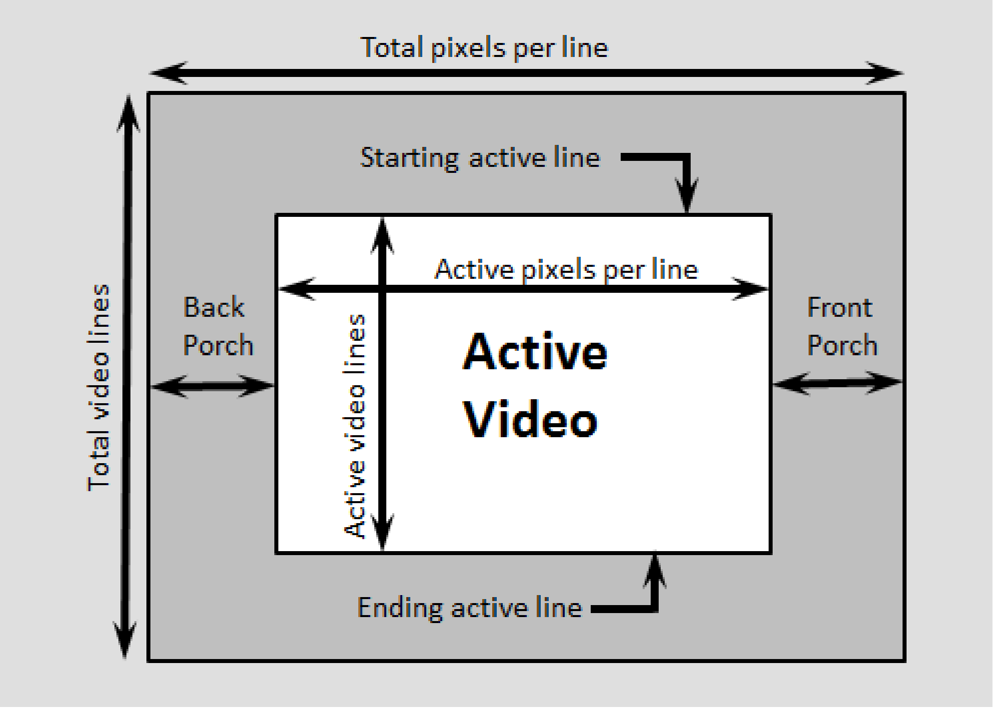
Size of the vector used to communicate with the FPGA subsystem. The block outputs pixels and control signals in vectors of the same length. The block calculates the length of the vectors based on the **Video format** parameter.

- `Pixel` — Output scalar values for pixel and control signals.
- `Line` — Output vectors contain **Total pixels per line** values.
- `Frame` — Output vectors contain **Total pixels per line** × **Total video lines** values.

A larger value results in faster communication between the FPGA board and Simulink. Choose the largest option that is supported by the I/O and memory resources on your board.

Video format

Dimensions of active and inactive regions of a video frame. To select a predefined format, use the **Video format** list. For a custom format, select `Custom`, and then specify the dimensions as integers. The frame dimensions are indicated in the diagram.



Note: The sample time of your video source must match the total number of pixels in the frame size you select in the Frame To Pixels block. Set the sample time to *Total pixels per line* × *Total lines*.

Video Format	Active Pixels Per Line	Active Video Lines	Total Pixels Per Line	Total Video Lines	Starting Active Line	Front Porch
240p	320	240	402	324	1	44
480p	640	480	800	525	36	16
480pH	720	480	858	525	33	16
576p	720	576	864	625	47	12
720p	1280	720	1650	750	25	110

Video Format	Active Pixels Per Line	Active Video Lines	Total Pixels Per Line	Total Video Lines	Starting Active Line	Front Porch
768p	1024	768	1344	806	10	24
1024p	1280	1024	1688	1066	42	48
1080p (default)	1920	1080	2200	1125	42	88
1200p	1600	1200	2160	1250	50	64
2KCinema	2048	1080	2750	1125	42	639
4KUHDTV	3840	2160	4400	2250	42	88
8KUHDTV	7680	4320	8800	4500	42	88
Custom	User-defined	User-defined	User-defined	User-defined	User-defined	User-defined

Note: When using a custom format, the values you enter for the active and inactive dimensions of the image must add up to the total frame dimensions.

For the horizontal direction, **Total pixels per line** must be greater than or equal to **Front porch** + **Active pixels per line**. The block calculates **Back porch** = **Total pixels per line** − **Front porch** − **Active pixels per line**.

For the vertical direction, **Total video lines** must be greater than or equal to **Starting active line** + **Active video lines** − 1. The block calculates **Ending active line** = **Starting active line** + **Active video lines** − 1.

If you specify a format that does not conform to these rules, the block reports an error.

Note: When using a custom format, **Active pixels per line** must be greater than 1. Also, set the horizontal blanking interval, or **Back porch** + **Front porch**, according to these guidelines.

- The total of **Back porch** + **Front porch** must be at least 2 times the largest *kernel size* of the algorithm in the blocks following the Frame To Pixel block. If the *kernel size* is < 4, the total porch must be at least 8 pixels.

- The **Back porch** must be at least 6 pixels. This parameter is the number of inactive pixels before the first valid pixel in a frame.

See Also

`FIL Pixels To Frame` | `Frame To Pixels`

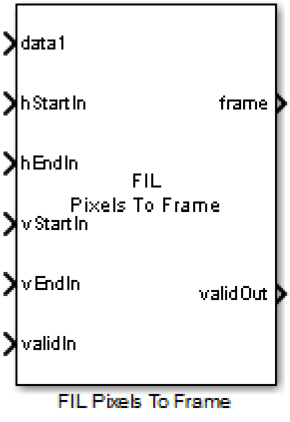
More About

- "Streaming Pixel Interface"
- "FPGA Verification"

Introduced in R2015a

FIL Pixels To Frame

Convert pixel stream from FPGA-in-the-loop to frame-based video



Library

visionhdlio

Description

The FIL Pixels To Frame block performs the same pixel-to-frame conversion as the `Pixels To Frame` block. In addition, you can configure the width of the input to be a single pixel, a line, or an entire frame per step. The block expects control signal input vectors of the same width as the pixel data. This optimization can speed up the communication link between the FPGA board and your Simulink simulation when using FPGA-in-the-loop. To run FPGA-in-the-loop, you must have an HDL Verifier license.

When you generate a programming file for a FIL target in Simulink, the tool creates a model to compare the FIL simulation with your Simulink design. For Vision HDL Toolbox designs, the FIL block in that model replicates the pixel-streaming interface to send one pixel at a time to the FPGA. You can modify the autogenerated model to use the FIL Frame To Pixels and FIL Pixels To Frame blocks to improve communication bandwidth with the FPGA board by sending one frame at a time. For how to modify the autogenerated model, see “FPGA-in-the-Loop”.

Specify the same video format for the FIL Frames To Pixels block and the FIL Pixels To Frame block.

Signal Attributes

Port	Direction	Description	Data Type
`data1,...,da`	Input	Image pixels, specified as a vector of *M* values, where *M* is the width of the **Output vector format**. There are *N* data ports, where *N* is the **Number of components**.	• `uint` or `int` • `fixdt()` • `boolean` • `double` or `single`
`hStartIn`	Input	Control signal indicating whether each pixel is the first pixel in a horizontal line of an input frame, returned as a vector of *M* values.	`boolean`
`hEndIn`	Input	Control signal indicating whether each pixel is the last pixel in a horizontal line of a frame, returned as a vector of *M* values.	`boolean`
`vStartIn`	Input	Control signal indicating whether each pixel is the first pixel in the first (top) line of a frame, returned as a vector of *M* values.	`boolean`
`vEndIn`	Input	Control signal indicating whether each pixel is the last pixel in the last (bottom) line of a frame, returned as a vector of *M* values.	`boolean`
`validIn`	Input	Control signal indicating the validity of the input pixel, returned as a vector of *M* values.	`boolean`
`matrix`	Output	Full image, returned as an **Active pixels per line**-by-**Active video lines**-by-*N* matrix. The height and width are the dimensions of the	Same as `data1,...,dataN`

Port	Direction	Description	Data Type
		active image specified in **Video format**. *N* is the **Number of components** used to express a single pixel.	
`validOut`	Output	True when the output frame is successfully recompiled from the input stream.	`boolean`

Parameters

Number of components

Component values of each pixel. The pixel can be represented by 1, 3, or 4 components. Set to 1 for grayscale video. Set to 3 for color video, for example, {R,G,B} or {Y,Cb,Cr}. Set to 4 to use color with an alpha channel for transparency. The output is an **Active pixels per line**-by-**Active video lines**-by-**Number of components** image matrix.

Input vector format

Size of the vector used to communicate with the FPGA subsystem. The block accepts input pixels and control signals in vectors of the same length. The block calculates the length of the vectors based on the **Video format** parameter.

- `Pixel` — Accept scalar values for pixel and control signals.
- `Line` — Accept input vectors containing **Total pixels per line** values.
- `Frame` — Accept input vectors containing **Total pixels per line** × **Total video lines** values.

A larger value results in faster communication between the FPGA board and Simulink. Choose the largest option that is supported by the I/O and memory resources on your board.

Video format

Dimensions of active and inactive regions of a video frame. To select a predefined format, use the **Video format** list. For a custom format, select `Custom`, and then specify the dimensions as integers.

Video Format	Active Pixels Per Line	Active Video Lines
240p	320	240
480p	640	480
480pH	720	480
576p	720	576
720p	1280	720
768p	1024	768
1024p	1280	1024
1080p (default)	1920	1080
1200p	1600	1200
2KCinema	2048	1080
4KUHDTV	3840	2160
8KUHDTV	7680	4320
Custom	User-defined	User-defined

See Also

FIL Frame To Pixels | Pixels To Frame

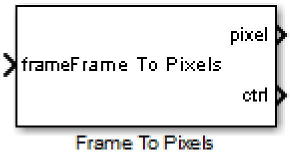
More About

- "Streaming Pixel Interface"
- "FPGA Verification"

Introduced in R2015a

Frame To Pixels

Convert frame-based video to pixel stream



Library

visionhdlio

Description

The Frame To Pixels block converts color or grayscale video frames to a pixel stream and control signals. The control signals indicate the validity of each pixel and its location in the frame. The pixel stream format can include padding pixels around the active frame. You can configure the frame and padding dimensions by selecting a common video format or specifying custom dimensions. See "Streaming Pixel Interface" for details of the pixel stream format.

Use this block to generate input for a subsystem targeted for HDL code generation. This block does not support HDL code generation.

If your model converts frames to a pixel stream and later converts the stream back to frames, specify the same video format for the Frame To Pixels block and the Pixels To Frame block.

Signal Attributes

Port	Direction	Description	Data Type
`frame`	Input	Full image specified as a **Active pixels per line**-by-**Active video**	• `uint` or `int` • `fixdt()`

Port	Direction	Description	Data Type
		lines-by-*N* matrix. Height and width are the dimensions of the active image specified in **Video format**. *N* is the **Number of components** used to express a single pixel.	• `boolean` • `double` or `single`
`pixel`	Output	Single image pixel returned as a vector of 1-by-**Number of components** values.	Specified by **Data type**
`ctrl`	Output	Control signals describing the validity of the pixel and the location of the pixel within the frame, specified as a bus containing five signals. See "Pixel Control Bus".	`pixelcontrol`

Parameters

Number of components

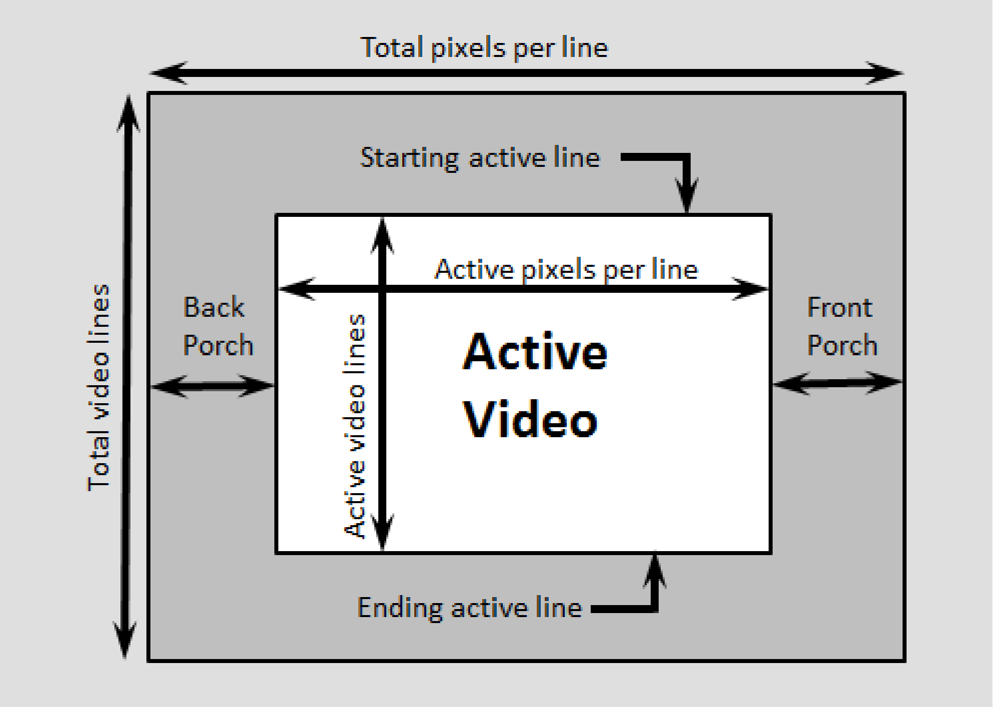
Component values of each pixel. The pixel can be represented by 1, 3, or 4 components. Set to 1 for grayscale video. Set to 3 for color video, for example, {R,G,B} or {Y,Cb,Cr}. Set to 4 to use color with an alpha channel for transparency. The output is an **Active pixels per line**-by-**Active video lines**-by-**Number of components** image matrix.

Data type

Output pixel data type. The default is `uint8`.

Video format

Dimensions of active and inactive regions of a video frame. To select a predefined format, use the **Video format** list. For a custom format, select `Custom`, and then specify the dimensions as integers. The frame dimensions are indicated in the diagram.



Note: The sample time of your video source must match the total number of pixels in the frame size you select in the Frame To Pixels block. Set the sample time to *Total pixels per line* × *Total lines*.

Video Format	Active Pixels Per Line	Active Video Lines	Total Pixels Per Line	Total Video Lines	Starting Active Line	Front Porch
240p	320	240	402	324	1	44
480p	640	480	800	525	36	16
480pH	720	480	858	525	33	16
576p	720	576	864	625	47	12
720p	1280	720	1650	750	25	110

Video Format	Active Pixels Per Line	Active Video Lines	Total Pixels Per Line	Total Video Lines	Starting Active Line	Front Porch
768p	1024	768	1344	806	10	24
1024p	1280	1024	1688	1066	42	48
1080p (default)	1920	1080	2200	1125	42	88
1200p	1600	1200	2160	1250	50	64
2KCinema	2048	1080	2750	1125	42	639
4KUHDTV	3840	2160	4400	2250	42	88
8KUHDTV	7680	4320	8800	4500	42	88
Custom	User-defined	User-defined	User-defined	User-defined	User-defined	User-defined

Note: When using a custom format, the values you enter for the active and inactive dimensions of the image must add up to the total frame dimensions.

For the horizontal direction, **Total pixels per line** must be greater than or equal to **Front porch** + **Active pixels per line**. The block calculates **Back porch** = **Total pixels per line** − **Front porch** − **Active pixels per line**.

For the vertical direction, **Total video lines** must be greater than or equal to **Starting active line** + **Active video lines** − 1. The block calculates **Ending active line** = **Starting active line** + **Active video lines** − 1.

If you specify a format that does not conform to these rules, the block reports an error.

Note: When using a custom format, **Active pixels per line** must be greater than 1. Also, set the horizontal blanking interval, or **Back porch** + **Front porch**, according to these guidelines.

- The total of **Back porch** + **Front porch** must be at least 2 times the largest *kernel size* of the algorithm in the blocks following the Frame To Pixel block. If the *kernel size* is < 4, the total porch must be at least 8 pixels.

- The **Back porch** must be at least 6 pixels. This parameter is the number of inactive pixels before the first valid pixel in a frame.

See Also

visionhdl.FrameToPixels | `Pixels To Frame`

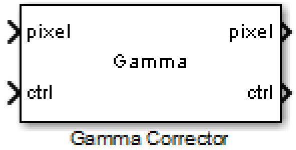
More About

- “Streaming Pixel Interface”

Introduced in R2015a

Gamma Corrector

Apply or remove gamma correction



Library

visionhdlconversions

Description

Gamma Corrector applies or removes gamma correction on a stream of pixels. Gamma correction adjusts linear pixel values so that the modified values fit a curve. The de-gamma operation performs the opposite operation to obtain linear pixel values.

This block uses a streaming pixel interface with a bus for synchronization control signals. This interface enables the block to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox blocks. The block accepts and returns a scalar pixel value and a bus containing five control signals. These signals indicate the validity of each pixel and the location of each pixel in the frame. To convert a pixel matrix into a pixel stream and these control signals, use the `Frame To Pixels` block. For a full description of the interface, see “Streaming Pixel Interface”.

Signal Attributes

Port	Direction	Description	Data Type
`pixel`	Input/ Output	Single image pixel, specified as a scalar value. The data type of the output is the same as the data type of the input.	• `uint8` or `uint16` • `int8` or `int16` • `fixdt(0,N,M)`, $N + M \leq$ 16

Port	Direction	Description	Data Type
			`double` and `single` data types are supported for simulation but not for HDL code generation.
`ctrl`	Input/ Output	Control signals describing the validity of the pixel and the location of the pixel within the frame, specified as a bus containing five signals. See "Pixel Control Bus".	`pixelcontrol`

Parameters

Operation

Direction of pixel value adjustment.

- `Gamma` (default) — Apply gamma correction.
- `De-gamma` — Remove gamma correction.

Gamma

Target gamma value, specified as a scalar greater than or equal to 1.

- When you set **Operation** to `Gamma`, specify **Gamma** as the target gamma value of the output video stream.
- When you set **Operation** to `De-gamma`, specify **Gamma** as the gamma value of the input video stream.

The default value is `2.2`.

Linear segment

Option to include a linear segment in the gamma curve. When you select this check box, the gamma curve has a linear portion near the origin. By default, this check box is selected.

Break point

Pixel value that corresponds to the point where the gamma curve and linear segment meet. Specify **Break point** as a scalar value between 0 and 1, exclusive. This parameter applies only when you select the **Linear segment** check box.

The default value is `0.018`.

Algorithm

For the equations used for gamma correction, see `Gamma Correction` in the Computer Vision System Toolbox documentation.

To save hardware resources, the block implements the gamma correction equation as a lookup table. The lookup table maps each input pixel value to a corrected output value.

Latency

The latency of the Gamma Corrector block is 2 cycles.

See Also

visionhdl.GammaCorrector | `Frame To Pixels` | `Gamma Correction`

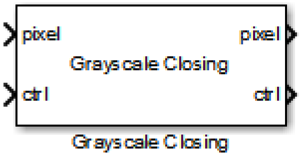
Related Examples

- "Gamma Correction"

Introduced in R2015a

Grayscale Closing

Morphological closing of grayscale pixel data



Library

visionhdlmorph

Description

The Grayscale Closing block performs morphological dilation, followed by morphological erosion, using the same neighborhood for both calculations. The block operates on a stream of pixel intensity values. You can specify a neighborhood, or structuring element, of up to 32×32 pixels. For line, square, or rectangle structuring elements more than 8 pixels wide, the block uses the Van Herk algorithm to find the maximum and minimum. For structuring elements less than 8 pixels wide, or that contain zero elements, the block implements a pipelined comparison tree to find the maximum and minimum.

This block uses a streaming pixel interface with a bus for synchronization control signals. This interface enables the block to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox blocks. The block accepts and returns a scalar pixel value and a bus containing five control signals. These signals indicate the validity of each pixel and the location of each pixel in the frame. To convert a pixel matrix into a pixel stream and these control signals, use the `Frame To Pixels` block. For a full description of the interface, see “Streaming Pixel Interface”.

Signal Attributes

Port	Direction	Description	Data Type
`pixel`	Input/ Output	Single image pixel, specified as a scalar value.	• `uint8`, `uint16`,`uint32`

Port	Direction	Description	Data Type
			• `fixdt(0,N,M)` `double` and `single` data types are supported for simulation but not for HDL code generation.
`ctrl`	Input/ Output	Control signals describing the validity of the pixel and the location of the pixel within the frame, specified as a bus containing five signals. See “Pixel Control Bus”.	`pixelcontrol`

Parameters

Neighborhood

Pixel neighborhood, specified as a matrix or vector of ones and zeros.

The block supports flat neighborhoods of up to 32×32 pixels. To use a structuring element, specify the **Neighborhood** as `getnhood(strel(`*`shape`*`))`. The minimum neighborhood size is a 2×2 matrix, or a 2×1 column vector. If the neighborhood is a row vector, it must be at least 8 columns wide and contain no zeros.

The default is `ones(3,3)`.

Line buffer size

Size of the line memory buffer, specified as a scalar integer.

Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the block uses the next largest power of two. The block allocates (*neighborhood lines* – 1)-by-**Line buffer size** memory locations to store the pixels. The default is 2048.

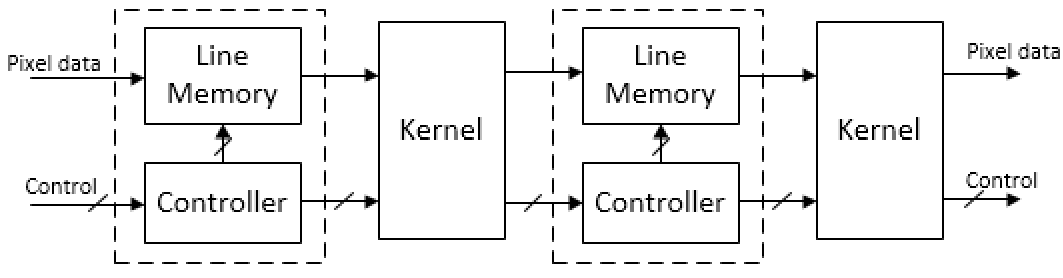
Algorithm

The closing algorithm is morphological dilation followed by morphological erosion. See the `Grayscale Dilation` and `Grayscale Erosion` reference pages for the respective kernel architectures.

The block pads the image with zeroes for the dilation operation, and with ones for the erosion operation. See "Edge Padding".

Latency

The total latency of the block is the line buffer latency plus the latency of the kernel calculation. The line buffer latency includes edge padding. Morphological closing is a compound operation. Therefore, this block contains a second line buffer between the dilation kernel and the erosion kernel. You can monitor the number of timesteps between the input control signals and the output control signals to determine the exact latency for any configuration of the block.



The latency of a Van Herk kernel for a neighborhood of $m \times n$ pixels is $2m + \log_2(n)$. The block implements this kernel for line, square, or rectangle structuring elements more than 8 pixels wide, with no pixels set to zero.

The latency of a comparison tree kernel for a neighborhood of $m \times n$ pixels is $\log_2(m)+\log_2(n)$.

Note: The horizontal blanking interval of the pixel stream format must be greater than the latency of the block. This interval enables the block to finish processing one line before it starts processing the next one. If you are using a custom video format, set the horizontal blanking interval using the `Frame To Pixels` block parameters. The horizontal blanking interval is equal to **Total pixels per line** – **Active pixels per line**, or, equivalently, **Front porch** + **Back porch**. Standard streaming video formats use a horizontal blanking interval of about 25% of the frame width. This interval is much larger than the filters applied to each frame.

See Also

visionhdl.GrayscaleClosing | `Closing` | `Frame To Pixels` | `Grayscale Dilation` | `Grayscale Erosion`

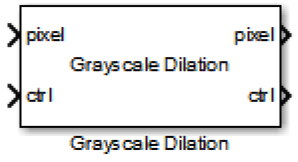
More About

- "Morphological Dilation and Erosion"
- "Structuring Elements"

Introduced in R2016a

Grayscale Dilation

Morphological dilation of grayscale pixel data



Library

visionhdlmorph

Description

The Grayscale Dilation block performs morphological dilation on a stream of pixel intensity values. You can specify a neighborhood, or structuring element, of up to 32×32 pixels. For line, square, or rectangle structuring elements more than 8 pixels wide, the block uses the Van Herk algorithm to find the maximum. This algorithm uses only three comparators to find the maximums of all the rows, then uses a comparison tree to find the maximum of the row results.

For structuring elements less than 8 pixels wide, or that contain zero elements, the block implements a pipelined comparison tree for each row of the neighborhood. An additional comparison tree finds the maximum value of the row results. If the structuring element contains zeros that mask off pixels, the algorithm saves hardware resources by not implementing comparators for those pixel locations.

This block uses a streaming pixel interface with a bus for synchronization control signals. This interface enables the block to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox blocks. The block accepts and returns a scalar pixel value and a bus containing five control signals. These signals indicate the validity of each pixel and the location of each pixel in the frame. To convert a pixel matrix into a pixel stream and these control signals, use the `Frame To Pixels` block. For a full description of the interface, see "Streaming Pixel Interface".

Signal Attributes

Port	Direction	Description	Data Type
`pixel`	Input/ Output	Single image pixel, specified as a scalar value.	• `uint8`, `uint16`,`uint32` • `fixdt(0,N,M)` `double` and `single` data types are supported for simulation but not for HDL code generation.
`ctrl`	Input/ Output	Control signals describing the validity of the pixel and the location of the pixel within the frame, specified as a bus containing five signals. See "Pixel Control Bus".	`pixelcontrol`

Parameters

Neighborhood

Pixel neighborhood, specified as a matrix or vector of ones and zeros.

The block supports flat neighborhoods of up to 32×32 pixels. To use a structuring element, specify the **Neighborhood** as `getnhood(strel(`*`shape`*`))`. The minimum neighborhood size is a 2×2 matrix, or a 2×1 column vector. If the neighborhood is a row vector, it must be at least 8 columns wide and contain no zeros.

The default is `ones(5,5)`.

Line buffer size

Size of the line memory buffer, specified as a scalar integer.

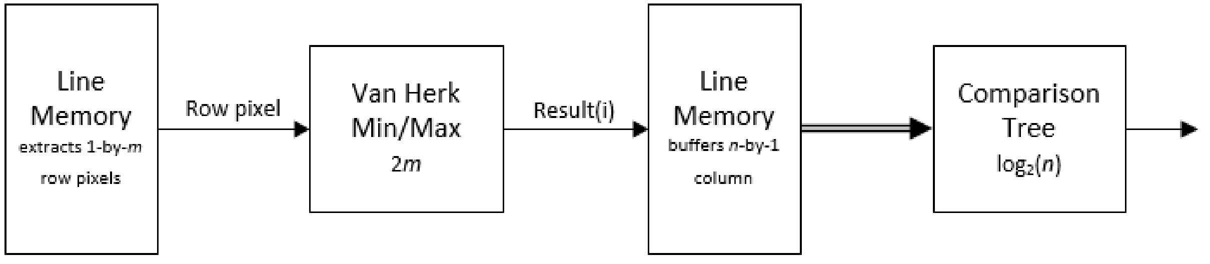
Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the block uses the next largest power of two. The block allocates (*neighborhood lines* – 1)-by-**Line buffer size** memory locations to store the pixels. The default is 2048.

Algorithm

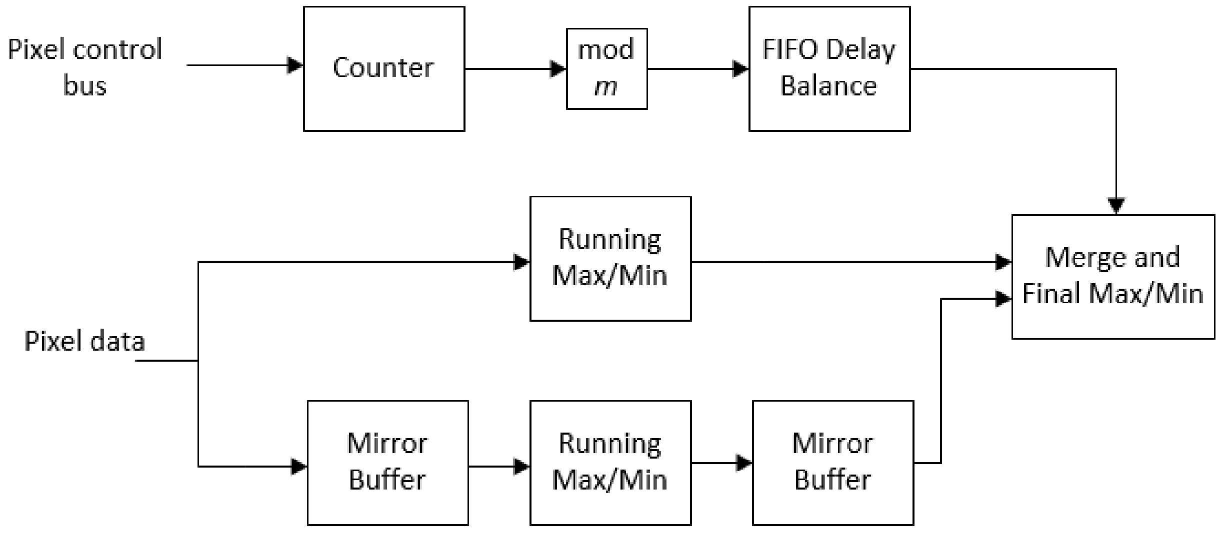
The block pads the image with zeroes for the dilation operation. See “Edge Padding”.

Van Herk Implementation

For line, square, or rectangle structuring elements more than 8 pixels wide, the block implements a Van Herk algorithm. All pixels in the structuring element must be set to one. The block decomposes the structuring element into rows and serially finds the maximum of each row using the Van Herk algorithm. If the size of the input frame is not a multiple of m pixels, the line memory also adds horizontal padding to a multiple of m. This implementation uses only 3 comparators total for all rows. Then, if there is more than one row, it calculates the maximum of the row results using a comparison tree. The diagram indicates the latency of each computation block.



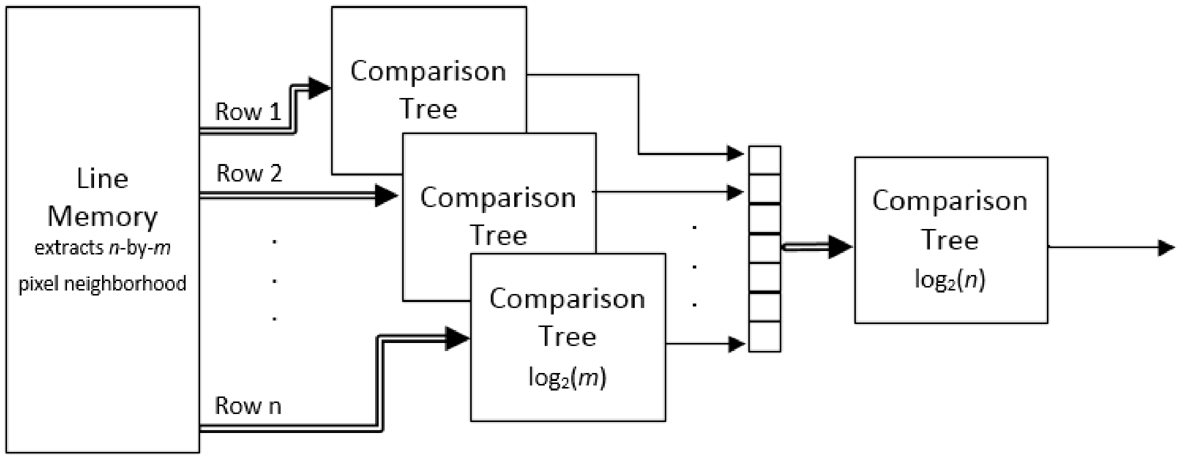
The Van Herk kernel computes a running forward maximum and a running backwards maximum on each row of the neighborhood. Therefore the pixels in the row must be buffered and the order reversed. The buffer adds latency relative to the comparison tree implementation. The Mirror Buffer is a ping-pong RAM of m pixels, where one memory reads values in reverse order while the other is writing. The kernel uses 3+n-1 comparators.



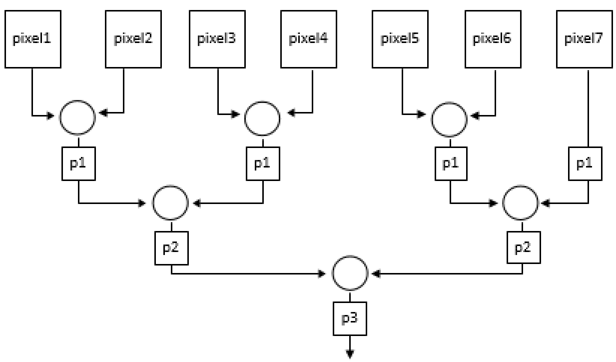
Comparison Tree Implementation

For structuring elements smaller than 8 pixels wide, or those with one or more pixels set to zero, the block implements a comparison tree.

The diagram shows the architecture of the dilation operation. The algorithm finds the maximum of each row of the neighborhood in parallel. Then it calculates the maximum of the rows using another comparison tree. The diagram indicates the latency of each computation block.

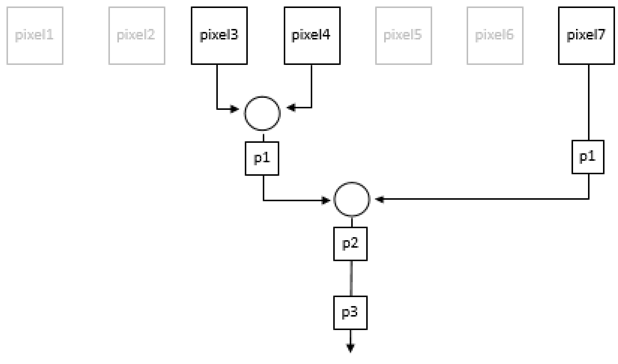


For a rectangular neighborhood that is m pixels wide, the first-stage comparison trees contain $m - 1$ comparators over $\log_2(m)$ clock cycles. For instance, for a rectangular neighborhood that is 7 pixels wide, the comparison tree has 6 comparators over 3 clock cycles.



However, if the neighborhood you specify contains zeroes, the generated HDL excludes the comparator for the zero locations. The pipeline delay through the comparison tree does not change. For instance, for a nonrectangular neighborhood with a row of `[0 0 1`

`1 0 0 1]`, the comparison tree for that row contains 2 comparators and still uses 3 clock cycles.

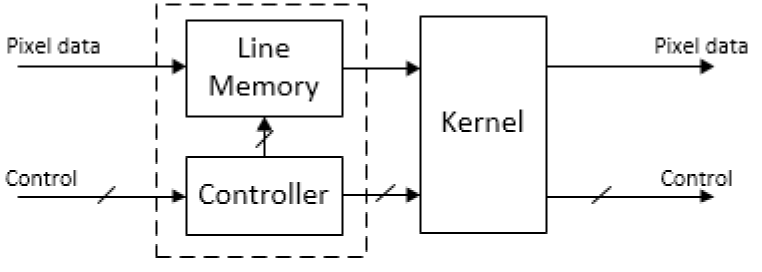


Latency

The latency of the operation is the line buffer latency plus the latency of the kernel calculation. The line buffer latency includes edge padding.

The latency of a Van Herk kernel for a neighborhood of $m \times n$ pixels is $2m + \log_2(n)$. The block implements this kernel for line, square, or rectangle structuring elements more than 8 pixels wide, with no pixels set to zero.

The latency of a comparison tree kernel for a neighborhood of $m \times n$ pixels is $\log_2(m)+\log_2(n)$. The block implements this kernel for structuring elements smaller than 8 pixels wide, or those with one or more pixels set to zero.



Note: The horizontal blanking interval of the pixel stream format must be greater than the latency of the block. This interval enables the block to finish processing one line before it starts processing the next one. If you are using a custom video format, set the horizontal blanking interval using the `Frame To Pixels` block parameters. The horizontal blanking interval is equal to **Total pixels per line** – **Active pixels per line**, or, equivalently, **Front porch** + **Back porch**. Standard streaming video formats use a horizontal blanking interval of about 25% of the frame width. This interval is much larger than the filters applied to each frame.

See Also

visionhdl.GrayscaleDilation | `Dilation` | `Frame To Pixels` | `Grayscale Erosion`

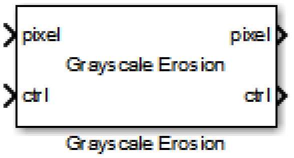
More About

- "Morphological Dilation and Erosion"
- "Structuring Elements"

Introduced in R2016a

Grayscale Erosion

Morphological erosion of grayscale pixel data



Library

visionhdlmorph

Description

The Grayscale Erosion block performs morphological erosion on a stream of pixel intensity values. You can specify a neighborhood, or structuring element, of up to 32×32 pixels. For line, square, or rectangle structuring elements more than 8 pixels wide, the block uses the Van Herk algorithm to find the minimum. This algorithm uses only three comparators to find the minimums of all the rows, then uses a comparison tree to find the minimum of the row results.

For structuring elements less than 8 pixels wide, or that contain zero elements, the block implements a pipelined comparison tree for each row of the neighborhood. An additional comparison tree finds the minimum value of the row results. If the structuring element contains zeros that mask off pixels, the algorithm saves hardware resources by not implementing comparators for those pixel locations.

This block uses a streaming pixel interface with a bus for synchronization control signals. This interface enables the block to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox blocks. The block accepts and returns a scalar pixel value and a bus containing five control signals. These signals indicate the validity of each pixel and the location of each pixel in the frame. To convert a pixel matrix into a pixel stream and these control signals, use the `Frame To Pixels` block. For a full description of the interface, see “Streaming Pixel Interface”.

Signal Attributes

Port	Direction	Description	Data Type
`pixel`	Input/ Output	Single image pixel, specified as a scalar value.	• `uint8`, `uint16`,`uint32` • `fixdt(0,N,M)` `double` and `single` data types are supported for simulation but not for HDL code generation.
`ctrl`	Input/ Output	Control signals describing the validity of the pixel and the location of the pixel within the frame, specified as a bus containing five signals. See "Pixel Control Bus".	`pixelcontrol`

Parameters

Neighborhood

Pixel neighborhood, specified as a matrix or vector of ones and zeros.

The block supports flat neighborhoods of up to 32×32 pixels. To use a structuring element, specify the **Neighborhood** as `getnhood(strel(`*`shape`*`))`. The minimum neighborhood size is a 2×2 matrix, or a 2×1 column vector. If the neighborhood is a row vector, it must be at least 8 columns wide and contain no zeros.

The default is `ones(3,3)`.

Line buffer size

Size of the line memory buffer, specified as a scalar integer.

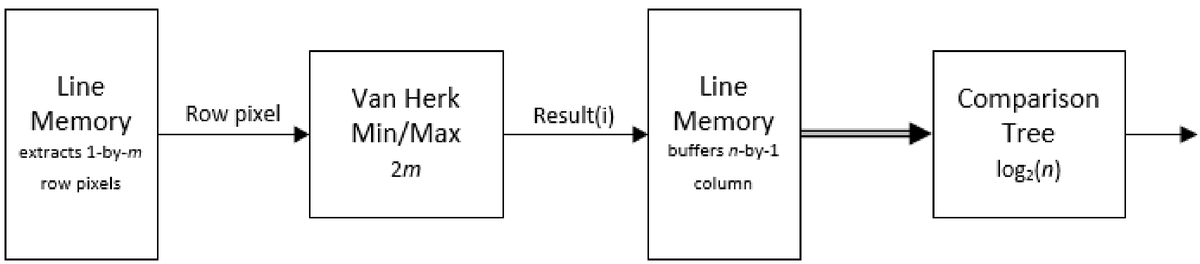
Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the block uses the next largest power of two. The block allocates (*neighborhood lines* – 1)-by-**Line buffer size** memory locations to store the pixels. The default is 2048.

Algorithm

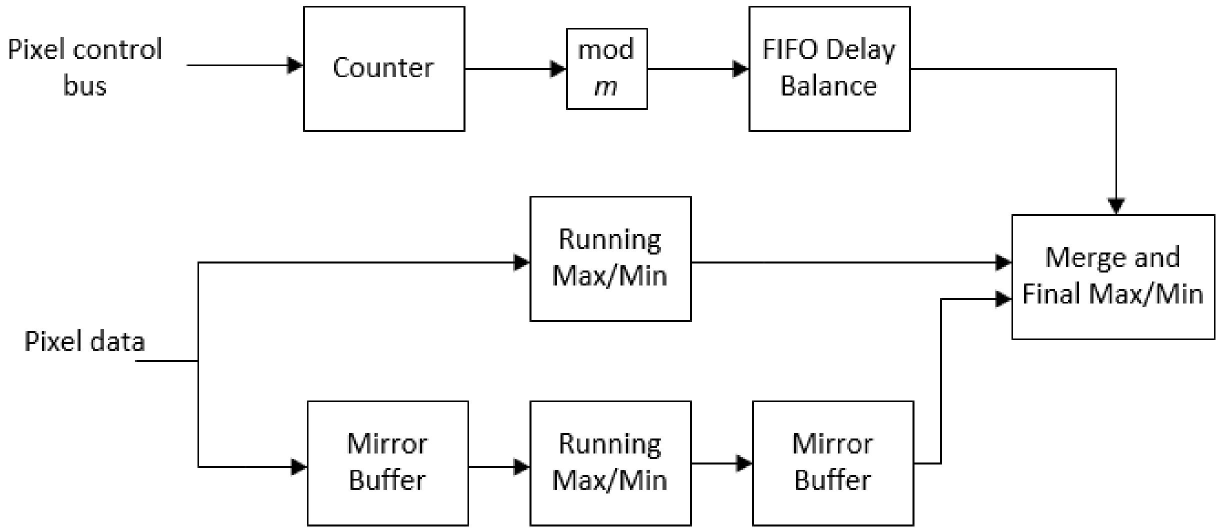
The block pads the image with ones for the erosion operation. See "Edge Padding".

Van Herk Implementation

For line, square, or rectangle structuring elements more than 8 pixels wide, the block implements a Van Herk algorithm. All pixels in the structuring element must be set to one. The block decomposes the structuring element into rows and serially finds the minimum of each row using the Van Herk algorithm. If the size of the input frame is not a multiple of *m* pixels, the line memory also adds horizontal padding to a multiple of *m*. This implementation uses only 3 comparators total for all rows. Then, if there is more than one row, it calculates the minimum of the row results using a comparison tree. The diagram indicates the latency of each computation block.



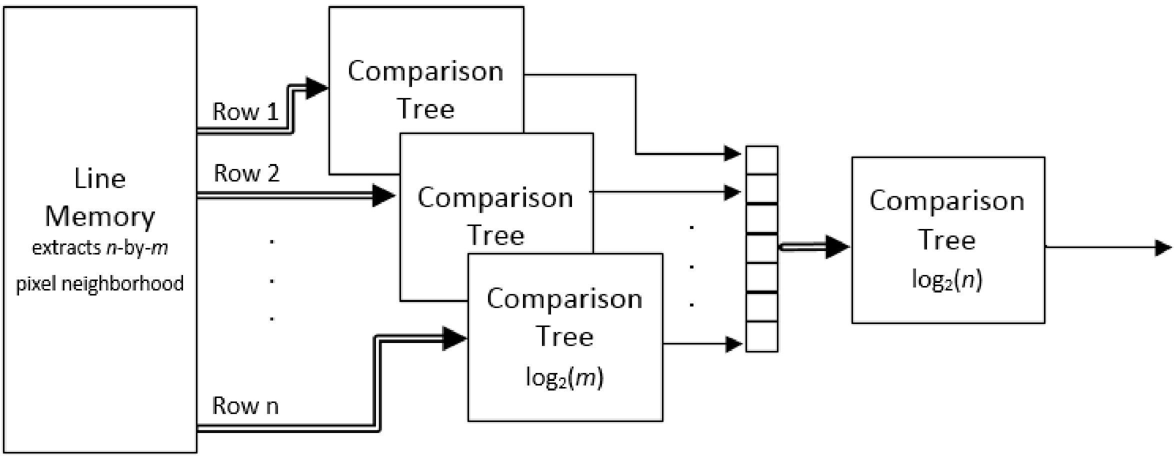
The Van Herk kernel computes a running forward minimum and a running backwards minimum on each row of the neighborhood. Therefore the pixels in the row must be buffered and the order reversed. The buffer adds latency relative to the comparison tree implementation. The Mirror Buffer is a ping-pong RAM of *m* pixels, where one memory reads values in reverse order while the other is writing. The kernel uses 3+*n*-1 comparators.



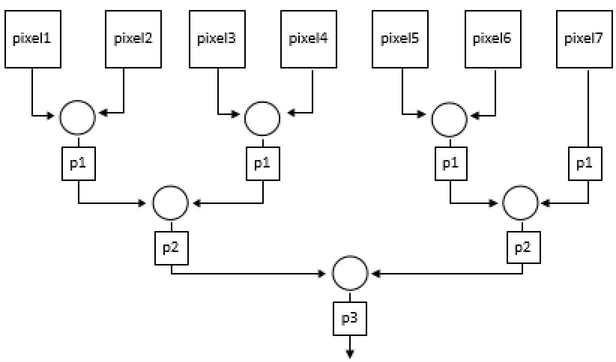
Comparison Tree Implementation

For structuring elements smaller than 8 pixels wide, or those with one or more pixels set to zero, the block implements a comparison tree.

The diagram shows the architecture of the erosion operation. The algorithm finds the minimum of each row of the neighborhood in parallel. Then it calculates the minimum of the rows using another comparison tree. The diagram indicates the latency of each computation block.

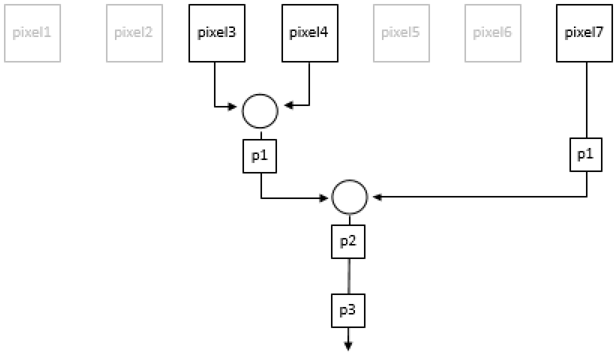


For a rectangular neighborhood that is *m* pixels wide, the first-stage comparison trees contain $m - 1$ comparators over $\log_2(m)$ clock cycles. For instance, for a rectangular neighborhood that is 7 pixels wide, the comparison tree has 6 comparators over 3 clock cycles.



However, if the neighborhood you specify contains zeroes, the generated HDL excludes the comparator for the zero locations. The pipeline delay through the comparison tree does not change. For instance, for a nonrectangular neighborhood with a row of `[0 0 1`

1 0 0 1], the comparison tree for that row contains 2 comparators and still uses 3 clock cycles.

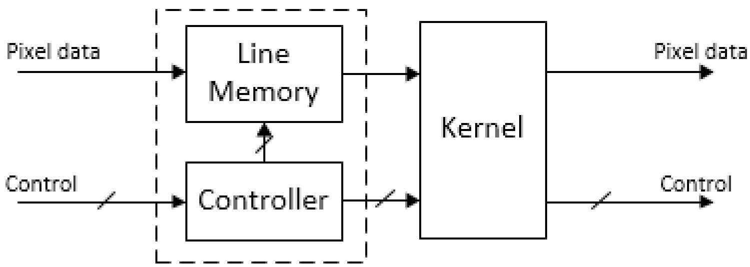


Latency

The latency of the operation is the line buffer latency plus the latency of the kernel calculation. The line buffer latency includes edge padding.

The latency of a Van Herk kernel for a neighborhood of $m \times n$ pixels is $2m + \log_2(n)$. The block implements this kernel for line, square, or rectangle structuring elements more than 8 pixels wide, with no pixels set to zero.

The latency of a comparison tree kernel for a neighborhood of $m \times n$ pixels is $\log_2(m)+\log_2(n)$. The block implements this kernel for structuring elements smaller than 8 pixels wide, or those with one or more pixels set to zero.



Note: The horizontal blanking interval of the pixel stream format must be greater than the latency of the block. This interval enables the block to finish processing one line before it starts processing the next one. If you are using a custom video format, set the horizontal blanking interval using the `Frame To Pixels` block parameters. The horizontal blanking interval is equal to **Total pixels per line** – **Active pixels per line**, or, equivalently, **Front porch** + **Back porch**. Standard streaming video formats use a horizontal blanking interval of about 25% of the frame width. This interval is much larger than the filters applied to each frame.

See Also

visionhdl.GrayscaleErosion | `Erosion` | `Frame To Pixels` | `Grayscale Dilation`

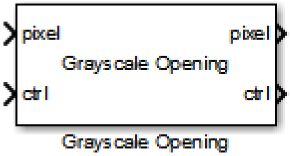
More About

- "Morphological Dilation and Erosion"
- "Structuring Elements"

Introduced in R2016a

Grayscale Opening

Morphological opening of grayscale pixel data



Library

visionhdlmorph

Description

The Grayscale Opening block performs morphological erosion, followed by morphological dilation, using the same neighborhood for both calculations. The block operates on a stream of pixel intensity values. You can specify a neighborhood, or structuring element, of up to 32×32 pixels. For line, square, or rectangle structuring elements more than 8 pixels wide, the block uses the Van Herk algorithm to find the maximum and minimum. For structuring elements less than 8 pixels wide, or that contain zero elements, the block implements a pipelined comparison tree to find the maximum and minimum.

This block uses a streaming pixel interface with a bus for synchronization control signals. This interface enables the block to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox blocks. The block accepts and returns a scalar pixel value and a bus containing five control signals. These signals indicate the validity of each pixel and the location of each pixel in the frame. To convert a pixel matrix into a pixel stream and these control signals, use the `Frame To Pixels` block. For a full description of the interface, see "Streaming Pixel Interface".

Signal Attributes

Port	Direction	Description	Data Type
`pixel`	Input/ Output	Single image pixel, specified as a scalar value.	• `uint8`, `uint16`,`uint32`

Port	Direction	Description	Data Type
			• `fixdt(0,N,M)` `double` and `single` data types are supported for simulation but not for HDL code generation.
`ctrl`	Input/ Output	Control signals describing the validity of the pixel and the location of the pixel within the frame, specified as a bus containing five signals. See "Pixel Control Bus".	`pixelcontrol`

Parameters

Neighborhood

Pixel neighborhood, specified as a matrix or vector of ones and zeros.

The block supports flat neighborhoods of up to 32×32 pixels. To use a structuring element, specify the **Neighborhood** as `getnhood(strel(`*`shape`*`))`. The minimum neighborhood size is a 2×2 matrix, or a 2×1 column vector. If the neighborhood is a row vector, it must be at least 8 columns wide and contain no zeros.

The default is `ones(3,3)`.

Line buffer size

Size of the line memory buffer, specified as a scalar integer.

Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the block uses the next largest power of two. The block allocates (*neighborhood lines* – 1)-by-**Line buffer size** memory locations to store the pixels. The default is 2048.

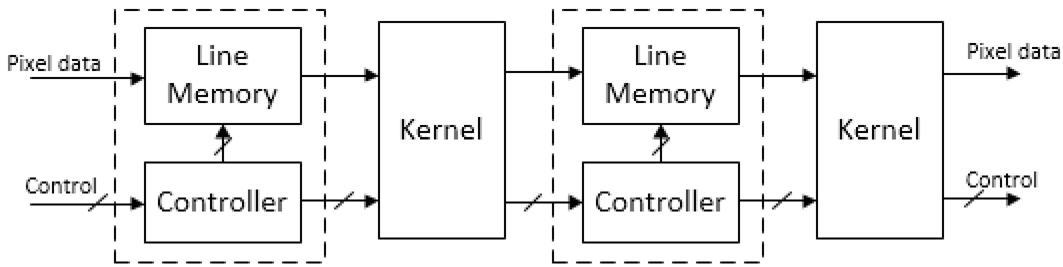
Algorithm

The opening algorithm is morphological erosion followed by morphological dilation. See the `Grayscale Erosion` and `Grayscale Dilation` reference pages for the respective kernel architectures.

The line memory pads the image with zeroes for the dilation operation, and with ones for the erosion operation. See “Edge Padding”.

Latency

The total latency of the block is the line buffer latency plus the latency of the kernel calculation. The latency of the line memory includes edge padding. Morphological opening is a compound operation. Therefore, this block contains a second line buffer between the erosion kernel and the dilation kernel. You can monitor the number of timesteps between the input control signals and the output control signals to determine the exact latency for any configuration of the block.



The latency of a Van Herk kernel for a neighborhood of $m \times n$ pixels is $2m + \log_2(n)$. The block implements this kernel for line, square, or rectangle structuring elements more than 8 pixels wide, with no pixels set to zero.

The latency of a comparison tree kernel for a neighborhood of $m \times n$ pixels is $\log_2(m)+\log_2(n)$.

Note: The horizontal blanking interval of the pixel stream format must be greater than the latency of the block. This interval enables the block to finish processing one line before it starts processing the next one. If you are using a custom video format, set the horizontal blanking interval using the `Frame To Pixels` block parameters. The horizontal blanking interval is equal to **Total pixels per line** – **Active pixels per line**, or, equivalently, **Front porch** + **Back porch**. Standard streaming video formats use a horizontal blanking interval of about 25% of the frame width. This interval is much larger than the filters applied to each frame.

See Also

visionhdl.GrayscaleOpening | `Frame To Pixels` | `Grayscale Dilation` | `Grayscale Erosion` | `Opening`

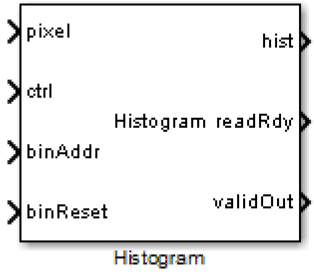
More About

- "Morphological Dilation and Erosion"
- "Structuring Elements"

Introduced in R2016a

Histogram

Frequency distribution



Library

visionhdlstatistics

Description

The Histogram block computes the frequency distribution of pixel values in a video stream. You can configure the number and size of the bins. The block provides a read interface for accessing each bin. The block keeps a running histogram until you reset the bin values.

This block uses a streaming pixel interface with a bus for synchronization control signals. This interface enables the block to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox blocks. The block accepts a scalar pixel value and a bus containing five control signals. These signals indicate the validity of each pixel and the location of each pixel in the frame. To convert a pixel matrix into a pixel stream and these control signals, use the `Frame To Pixels` block. For a full description of the interface, see “Streaming Pixel Interface”.

Signal Attributes

Port	Direction	Description	Data Type
`pixel`	Input	Single image pixel, specified as an unsigned integer scalar.	• `uint` • `fixdt(0,N,0)` `double` and `single` data types are supported for simulation but not for HDL code generation.
`ctrl`	Input	Control signals describing the validity of the pixel and the location of the pixel within the frame, specified as a bus containing five signals. See “Pixel Control Bus”.	`pixelcontrol`
`binAddr`	Input	Bin number for reading histogram values. The block captures this value each cycle that `readRdy` is `true`.	`fixdt(0,N,0)`, N = 5,6,...,10. Word length must be $\log_2$(**Number of bins**).
`binReset`	Input	Triggers RAM initialization sequence when `true`.	`boolean`
`readRdy`	Output	Indicates `true` when histogram is ready for read.	`boolean`
`hist`	Output	Histogram value corresponding to a `binAddr` request, returned as a scalar.	`fixdt(0,N,0)` `double` and `single` data types are supported for simulation but not for HDL code generation.
`validOut`	Output	Indicates `true` when `hist` is available.	`boolean`

Parameters

Number of bins

Number of bins for the histogram.

Choose the number of bins depending on the input word length (WL). If the number of bins is less than 2^{WL}, the block truncates the least-significant bits of each pixel. If the number of bins is greater than 2^{WL}, the block warns about an inefficient use of hardware resources. The default is 256.

Data type

Data type of the histogram bin values.

- `double`
- `single`
- `Unsigned fixed point` (default)

`double` and `single` data types are supported for simulation but not for HDL code generation.

Word length

Word length of the histogram bins when **Data type** is `Unsigned fixed point`. If a bin overflows, the count saturates and the block shows a warning. The default is 16.

Algorithm

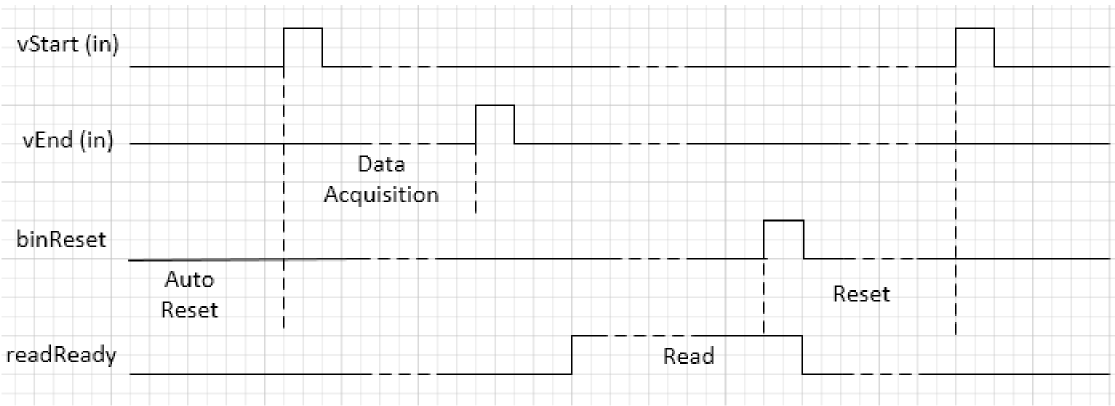
RAM Reset and Ready Sequence

At startup, you must wait **Number of bins** cycles for the block to reset the RAM, before sending input data. This initial reset happens without asserting `binReset`.

You cannot read histogram bins and apply pixel data at the same time. When you want to read the bin values, wait for `readRdy` and then apply each bin address of interest. The block provides the corresponding histogram values on the `hist` port, with accompanying `validOut` signal.

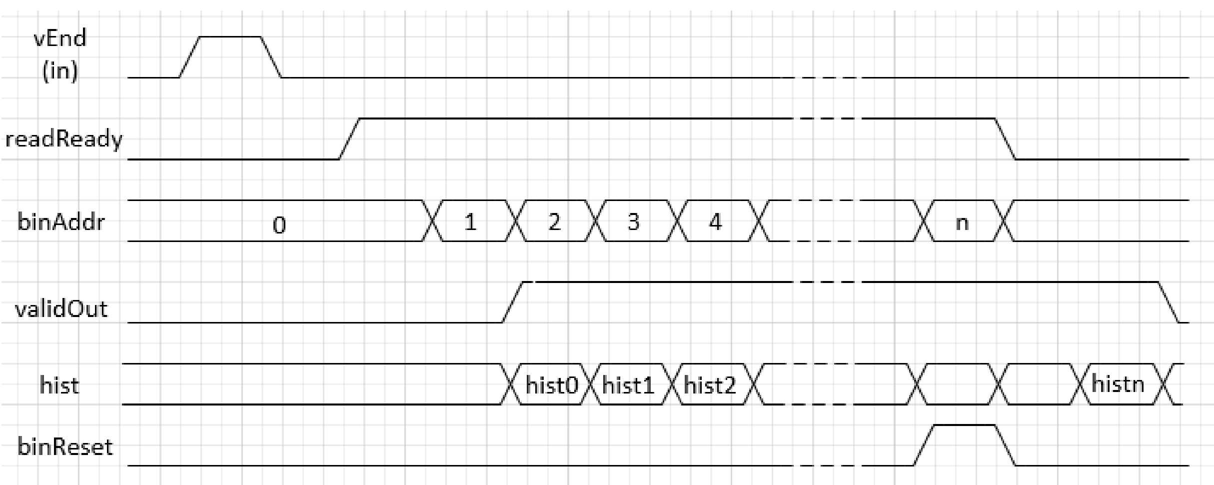
The histogram values persist and accumulate across frames until you assert `binReset`. When you assert `binReset`, the block takes **Number of bins** cycles to clear the RAM and be ready for new input. Other input signals are ignored during reset.

The diagram shows an overview of the reset sequence. `vStart` and `vEnd` are control signals in the `pixelcontrol` input bus.



The diagram shows the automatic startup reset, followed by a frame of video input. The read window starts when `readReady` is asserted. The `binReset` signal initiates a bin reset. The next input frame is not applied until after the reset is complete.

The diagram illustrates a bin read sequence. `vEnd` is a control signal in the `pixelcontrol` input bus. `validOut` indicates when the bin values on `hist` are available.



After the last pixel of a video frame, indicated by `vEnd` = `true`, the block asserts `readRdy` to show that the histogram is ready for reading. Two cycles after applying a bin address, the block provides the value of that bin on hist, with a corresponding valid signal. You can request the last bin address and assert binReset at the same time.

Latency

The block sets readRdy to true 2 cycles after receiving the last pixel of a frame. The input pixelcontrol bus indicates the last pixel of a frame by vEnd = true. While readRdy is true, the block captures binAddr requests on each cycle. The block provides the corresponding histogram bin values on hist two cycles later.

See Also

visionhdl.Histogram | 2-D Histogram | Frame To Pixels | imhist

Related Examples

- “Histogram Equalization”

Introduced in R2015a

Image Filter

2-D FIR filtering

Library

visionhdlfilter

Description

The Image Filter block performs two-dimensional FIR filtering on a pixel stream.

This block uses a streaming pixel interface with a bus for synchronization control signals. This interface enables the block to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox blocks. The block accepts and returns a scalar pixel value and a bus containing five control signals. These signals indicate the validity of each pixel and the location of each pixel in the frame. To convert a pixel matrix into a pixel stream and these control signals, use the `Frame To Pixels` block. For a full description of the interface, see "Streaming Pixel Interface".

Signal Attributes

Port	Direction	Description	Data Type
`pixel`	Input	Single pixel, specified as a scalar value.	• `uint` or `int` • `fixdt()` `double` and `single` data types are supported for simulation but not for HDL code generation.

Port	Direction	Description	Data Type
ctrl	Input/ Output	Control signals describing the validity of the pixel and the location of the pixel within the frame, specified as a bus containing five signals. See "Pixel Control Bus".	pixelcontrol
pixel	Output	Single pixel, returned as a scalar value. You can specify the output data type in the block dialog box.	• uint or int • fixdt() double and single data types are supported for simulation but not for HDL code generation.

Parameters

Main

Filter coefficients

Coefficients of the filter, specified as a vector or matrix of any numeric type.

The maximum size along any dimension of a matrix or vector is 16.

Padding method

Method for padding the boundary of the input image. See "Edge Padding".

- Constant (default) — Interpret pixels outside the image frame as having a constant value.
- Replicate — Repeat the value of pixels at the edge of the image.
- Symmetric — Pad the input matrix with its mirror image.

Padding value

Constant value used to pad the boundary of the input image.

This parameter is visible when you set **Padding method** to Constant. The block casts this value to the same data type as the input pixel. The default value is 0.

Line buffer size

Size of the line memory buffer, specified as a scalar integer.

Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the block uses the next largest power of two. The block allocates (*coefficient rows – 1*)-by-**Line buffer size** memory locations to store the pixels. The default value is 2048.

Data Types

Rounding mode

Rounding mode for internal fixed-point calculations. The default is `Floor`.

Overflow mode

Overflow mode for internal fixed-point calculations. The default is `Wrap`.

Coefficients Data Type

Method for determining the data type of the filter coefficients.

The default is `Inherit: Same as first input`.

When converting the coefficients to this data type, the block always uses `Saturate` overflow mode and `Nearest` rounding mode.

Output Data Type

Method for determining the data type of the output pixels.

The default is `Inherit: Same as first input`.

Lock data type settings against changes by the fixed-point tools

Select to lock all data type settings of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Lock the Output Data Type Setting" in the Fixed-Point Designer™ documentation.

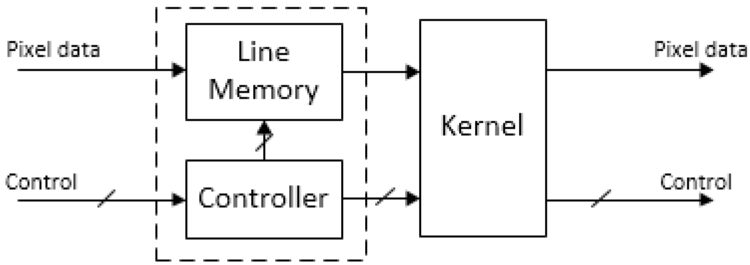
Algorithm

The block implements the filter with a fully-pipelined architecture. Each multiplier has two pipeline stages on each input and two pipeline stages on each output. The adder is a pipelined tree structure. HDL code generation takes advantage of symmetric, unity, or zero-value coefficients to reduce the number of multipliers.

You can optimize the multipliers for HDL code generation using canonical signed digit (CSD) or factored CSD. Right-click the block, select **HDL Code** > **HDL Properties**, and set the **ConstMultiplierOptimization** parameter to `csd` or `fcsd`.

Latency

The latency of the block is the line buffer latency plus the latency of the kernel calculation. The line buffer latency includes edge padding. To determine the exact latency for any configuration of the block, you can measure the number of time steps between the input control signals and the output control signals .



The latency of the kernel varies depending on the coefficients you choose.

Note: The horizontal blanking interval of the pixel stream format must be greater than the latency of the block. This interval enables the block to finish processing one line before it starts processing the next one. If you are using a custom video format, set the horizontal blanking interval using the `Frame To Pixels` block parameters. The horizontal blanking interval is equal to **Total pixels per line** – **Active pixels per line**, or, equivalently, **Front porch** + **Back porch**. Standard streaming video formats use a horizontal blanking interval of about 25% of the frame width. This interval is much larger than the filters applied to each frame.

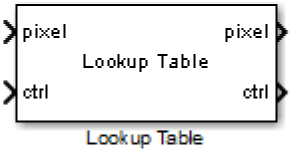
See Also

visionhdl.ImageFilter | `2-D FIR Filter` | `Frame To Pixels`

Introduced in R2015a

Lookup Table

Map input pixel to output pixel using custom rule



Library

visionhdlconversions

Description

The Lookup Table block provides a custom one-to-one map between input pixel values and output pixel values.

This block uses a streaming pixel interface with a bus for synchronization control signals. This interface enables the block to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox blocks. The block accepts and returns a scalar pixel value and a bus containing five control signals. These signals indicate the validity of each pixel and the location of each pixel in the frame. To convert a pixel matrix into a pixel stream and these control signals, use the `Frame To Pixels` block. For a full description of the interface, see "Streaming Pixel Interface".

Signal Attributes

Port	Direction	Description	Data Type
`pixel`	Input	Single image pixel, specified as a scalar value.	• `boolean` • `uint8` or `uint16` • `fixdt(0,N,M)`, $N + M \leq 16$

Port	Direction	Description	Data Type
`ctrl`	Input/Output	Control signals describing the validity of the pixel and the location of the pixel within the frame, specified as a bus containing five signals. See “Pixel Control Bus”.	`pixelcontrol`
`pixel`	Output	Single image pixel, returned as a scalar value.	Specified by **Table data**. `double` and `single` data types are supported for simulation but not for HDL code generation.

Parameters

Table data

Determines the one-to-one correspondence between an input `pixel` value and an output `pixel` value.

- The table data is a row or column vector of any data type. The data type of the table data determines that of the output `pixel`.
- The length of the vector must be $2^{WordLength}$, where *WordLength* is the size, in bits, of the input `pixel`.
- The smallest representable value of the input data type maps to the first element of the table, the second smallest value maps to the second element, and so on. For example, if the input `pixel` has a data type of `fixdt(0,3,1)`, the input value 0 maps to the first element of the table, 0.5 maps to the second element, 1 maps to the third, and so on.

The default value is `uint8(0:1:255)`.

Algorithm

Latency

The latency of the Lookup Table block is 2 cycles.

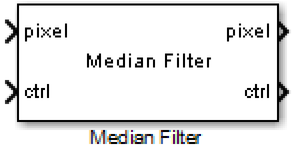
See Also

visionhdl.LookupTable | `Frame To Pixels`

Introduced in R2015a

Median Filter

2-D median filtering



Library

visionhdlfilter

Description

Median Filter replaces each pixel with the median value of the surrounding *N*-by-*N* neighborhood. The median is less sensitive to extreme values than the mean. Use this block to remove salt-and-pepper noise from an image without significantly reducing the sharpness of the image. You can specify the neighborhood size and the padding values for the edges of the input image.

This block uses a streaming pixel interface with a bus for synchronization control signals. This interface enables the block to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox blocks. The block accepts and returns a scalar pixel value and a bus containing five control signals. These signals indicate the validity of each pixel and the location of each pixel in the frame. To convert a pixel matrix into a pixel stream and these control signals, use the `Frame To Pixels` block. For a full description of the interface, see "Streaming Pixel Interface".

Signal Attributes

Port	Direction	Description	Data Type
`pixel`	Input/Output	Single image pixel, specified as a scalar integer value. The data	• `uint` or `int` • `fixdt(~,N,0)`

Port	Direction	Description	Data Type
		type of the output is the same as the data type of the input.	• boolean double and single data types are supported for simulation but not for HDL code generation.
ctrl	Input/Output	Control signals describing the validity of the pixel and the location of the pixel within the frame, specified as a bus containing five signals. See "Pixel Control Bus".	pixelcontrol

Parameters

Neighborhood size

Size in pixels of the image region used to compute the median.

- 3×3 (default)
- 5×5
- 7×7

Padding method

Method for padding the boundary of the input image. See "Edge Padding".

- Constant — Pad input matrix with a constant value.
- Replicate — Repeat the value of pixels at the edge of the image.
- Symmetric (default) — Pad image edge with its mirror image.

Padding value

Constant value used to pad the boundary of the input image.

This parameter is visible when you set **Padding method** to Constant. The block casts this value to the same data type as the input pixel. The default value is 0.

Line buffer size

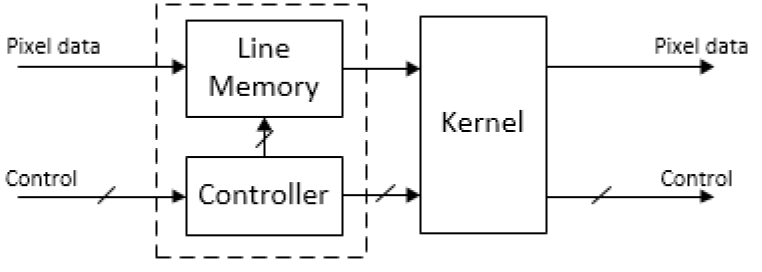
Size of the line memory buffer, specified as a scalar integer.

Choose a power of 2 that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the object uses the next largest power of two. The block allocates *N - 1*-by-**Line buffer size** memory locations to store the pixels used to compute the median value. *N* is the dimension of the square region specified in **Neighborhood size**. The default value is 2048.

Algorithm

Latency

The latency of the block is the line buffer latency plus the latency of the kernel calculation. The line buffer latency includes edge padding. To determine the exact latency for any configuration of the block, you can measure the number of time steps between the input control signals and the output control signals .



The latency of the filter kernel depends on the neighborhood size as shown in the table.

Neighborhood size	# of Comparisons to Find Median
3×3	11
5×5	75
7×7	230

Note: The horizontal blanking interval of the pixel stream format must be greater than the latency of the block. This interval enables the block to finish processing one line before it starts processing the next one. If you are using a custom video format, set the horizontal blanking interval using the `Frame To Pixels` block parameters. The horizontal blanking interval is equal to **Total pixels per line** – **Active pixels per line**, or, equivalently, **Front porch** + **Back porch**. Standard streaming video formats use a horizontal blanking interval of about 25% of the frame width. This interval is much larger than the filters applied to each frame.

See Also

visionhdl.MedianFilter | `Frame To Pixels` | `Median Filter`

Introduced in R2015a

Opening

Morphological opening of binary pixel data

Library

visionhdlmorph

Description

The Opening block performs morphological erosion, followed by morphological dilation, using the same neighborhood for both calculations. The block operates on a stream of binary intensity values.

This block uses a streaming pixel interface with a bus for synchronization control signals. This interface enables the block to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox blocks. The block accepts and returns a scalar pixel value and a bus containing five control signals. These signals indicate the validity of each pixel and the location of each pixel in the frame. To convert a pixel matrix into a pixel stream and these control signals, use the `Frame To Pixels` block. For a full description of the interface, see “Streaming Pixel Interface”.

Signal Attributes

Port	Direction	Description	Data Type
`pixel`	Input/ Output	Single image pixel, specified as a scalar binary value.	`boolean`
`ctrl`	Input/ Output	Control signals describing the validity of the pixel and the	`pixelcontrol`

Port	Direction	Description	Data Type
		location of the pixel within the frame, specified as a bus containing five signals. See “Pixel Control Bus”.	

Parameters

Neighborhood

Pixel neighborhood, specified as a matrix or vector of ones and zeros.

The block supports neighborhoods of up to 32×32 pixels. To use a structuring element, specify **Neighborhood** as `getnhood(strel(`*`shape`*`))`.

The default is `[0,1,0;1,1,1;0,1,0]`.

Line buffer size

Size of the line memory buffer, specified as a scalar integer.

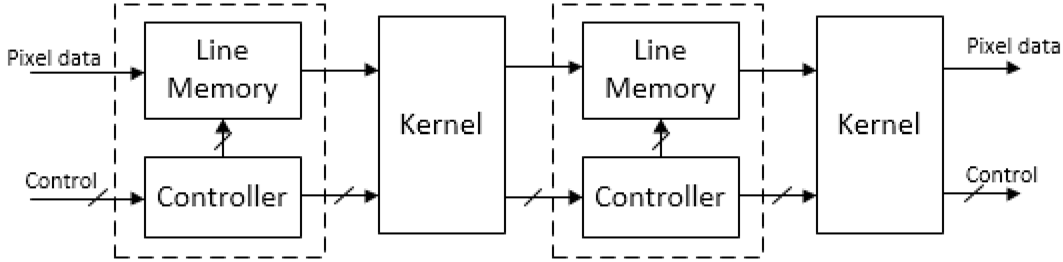
Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the block uses the next largest power of two. The block allocates (*neighborhood lines* – 1)-by-**Line buffer size** memory locations to store the pixels. The default is 2048.

Algorithm

The block pads the image with zeroes for the dilation operation, and with ones for the erosion operation. See “Edge Padding”.

Latency

The total latency of the block is the line buffer latency plus the latency of the kernel calculation. Morphological opening is a compound operation. Therefore, this block contains a second line buffer between the erosion kernel and the dilation kernel. To determine the exact latency for any configuration of the block, monitor the number of time steps between the input control signals and the output control signals.



The latency of the line memory includes edge padding. The latency of the kernel depends on the neighborhood size.

Note: The horizontal blanking interval of the pixel stream format must be greater than the latency of the block. This interval enables the block to finish processing one line before it starts processing the next one. If you are using a custom video format, set the horizontal blanking interval using the `Frame To Pixels` block parameters. The horizontal blanking interval is equal to **Total pixels per line** – **Active pixels per line**, or, equivalently, **Front porch** + **Back porch**. Standard streaming video formats use a horizontal blanking interval of about 25% of the frame width. This interval is much larger than the filters applied to each frame.

See Also

visionhdl.Opening | `Closing` | `Dilation` | `Erosion` | `Frame To Pixels` | `Opening`

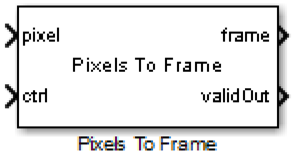
More About

- "Morphological Dilation and Erosion"
- "Structuring Elements"

Introduced in R2015a

Pixels To Frame

Convert pixel stream to frame-based video



Library

visionhdlio

Description

The Pixels To Frame block converts a color or grayscale pixel stream and control signals to frame-based video. The control signal bus indicates the validity of each pixel and its location within the frame. The pixel stream format can include padding pixels around the active frame. You can configure the frame and padding dimensions by selecting a common video format or specifying custom dimensions. See "Streaming Pixel Interface" for details of the pixel stream format.

Use this block to convert the output of a subsystem targeted for HDL code generation back to frames. This block does not support HDL code generation.

If your model converts frames to a pixel stream and later converts the stream back to frames, specify the same video format for the Frame To Pixels block and the Pixels To Frame block.

Signal Attributes

The Pixels To Frame block has the following input and output ports.

Port	Direction	Description	Data Type
`pixel`	Input	Single image pixel specified by a vector of 1-by-**Number of components** values.	• `uint` or `int` • `fixdt()`

Port	Direction	Description	Data Type
			• boolean • double or single
ctrl	Input	Control signals describing the validity of the pixel and the location of the pixel within the frame, specified as a bus containing five signals. See "Pixel Control Bus".	pixelcontrol
frame	Output	Full image returned as a **Active pixels per line**-by-**Active video lines**-by-*N* matrix. Height and width are the dimensions of the active image specified in **Video format**. *N* is the **Number of components** used to express a single pixel.	Same as pixel
validOut	Output	True when the output frame is successfully recompiled from the input stream.	boolean

Parameters

Number of components

Component values of each pixel. The pixel can be represented by 1, 3, or 4 components. Set to 1 for grayscale video. Set to 3 for color video, for example, {R,G,B} or {Y,Cb,Cr}. Set to 4 to use color with an alpha channel for transparency. The output is an **Active pixels per line**-by-**Active video lines**-by-**Number of components** image matrix.

Video format

Dimensions of active and inactive regions of a video frame. To select a predefined format, use the **Video format** pull-down menu. For a custom format, select Custom, then specify the dimensions as integers.

Video Format	Active Pixels Per Line	Active Video Lines
240p	320	240

Video Format	Active Pixels Per Line	Active Video Lines
480p	640	480
480pH	720	480
576p	720	576
720p	1280	720
768p	1024	768
1024p	1280	1024
1080p (default)	1920	1080
1200p	1600	1200
2KCinema	2048	1080
4KUHDTV	3840	2160
8KUHDTV	7680	4320
Custom	User-defined	User-defined

See Also

visionhdl.PixelsToFrame | Frame To Pixels

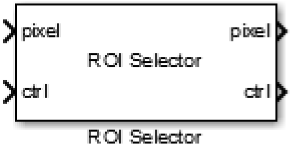
More About

- "Streaming Pixel Interface"

Introduced in R2015a

ROI Selector

Select a region of interest (ROI) from pixel stream

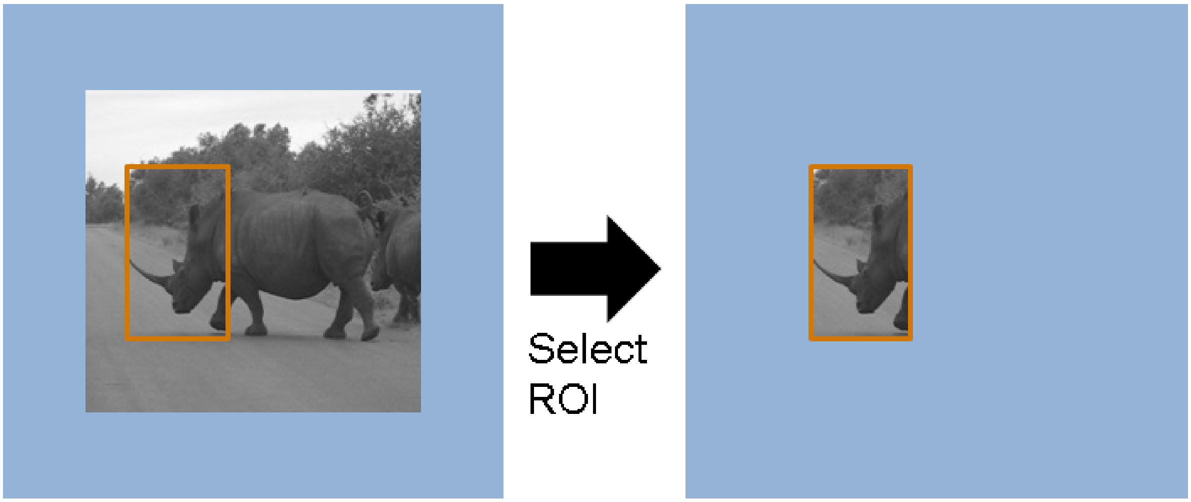


Library

visionhdlutilities

Description

The ROI Selector block selects a portion of the active frame from a video stream. The total size of the frame remains the same. The control signals indicate a new active region of the frame. The diagram shows the inactive pixel regions in blue and the requested output region outlined in orange.



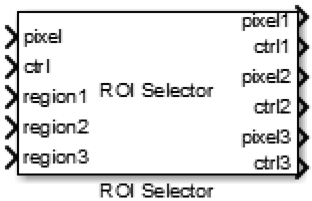
You can specify a fixed size and location for the new frame, or select the frame location in real time via an input port. You can select more than one region. Define each region by the upper-left corner coordinates and the dimensions. The block returns one set of pixels and control signals for each region you specify. The block sets the inactive pixels in the output frame to zero.

Regions are independent from each other, so they can overlap. If you specify a region that includes the edge of the active frame, the block returns only the active portion of the region. The diagram shows the output frames for three requested regions. The second output region (treetops) does not include the inactive region above the frame.

This block uses a streaming pixel interface with a bus for synchronization control signals. This interface enables the block to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox blocks. The block accepts and returns a scalar pixel value and a bus containing five control signals. These signals indicate the validity of each pixel and the location of each pixel in the frame. To convert a pixel matrix into a pixel stream and these control signals, use the `Frame To Pixels` block. For a full description of the interface, see "Streaming Pixel Interface".

Signal Attributes

The diagram shows the additional ports on the block when you select three regions. The `regionN` input ports are optional.



Port	Direction	Description	Data Type
`pixel`	Input	Single image pixel, specified as a scalar value.	Any `numeric` data type `double` and `single` data types are supported for simulation but not for HDL code generation.
`ctrl`	Input	Control signals describing the validity of the pixel and the location of the pixel within the frame, specified as a bus containing five signals. See "Pixel Control Bus".	`pixelcontrol`

Port	Direction	Description	Data Type
`region1,...,regio`	Input	Region of interest, specified as vectors of positive integers that define the coordinates of the top-left corner, and the dimensions, of each desired output frame, `[hPos vPos hSize vSize]`. The block has *N* region ports, where *N* is the **Number of regions**.	1-by-4 vector of positive integers
`pixel1,...,pixelN`	Output	Output image pixels, specified as scalar values. The block has *N* output pixel ports, where *N* is the **Number of regions**, or the size of the **Regions** matrix.	Same data type as the input `pixel` port
`ctrl1,...,ctrlN`	Output	Control signals, specifies as busses of five signals each. The block has *N* output control ports, where *N* is the **Number of regions**, or the size of the **Regions** matrix.	`pixelcontrol`

Parameters

Regions source

Location of the output region definitions

Select `Property` to specify the region(s) in the **Regions** mask parameter. Select `Input port` to specify the region(s) on input ports. There is one input port for each region. The block samples the `region` input ports when `vStart` is set in the input control bus.

Regions

Rectangular regions of interest to select from the input frame, specified as a *N*-by-4 matrix.

N is the number of regions. You can select up to 16 regions. The four elements that define each region are the top-left starting coordinates and the dimensions, `[hPos vPos hSize vSize]`. The coordinates count from the upper left corner of the active frame, defined as [1,1]. `hSize` must be greater than 1. The regions are independent of each other, so they can overlap. This parameter applies when you set **Regions source** to `Property`.

Number of regions

Number of region input ports, specified as an integer from 1 to 16.

This parameter applies when you set **Regions source** to `Input port`.

Algorithm

The generated HDL code for the ROI Selector block uses two 32-bit counters. It does not use additional counters for additional regions.

Latency

The block has a latency of three cycles.

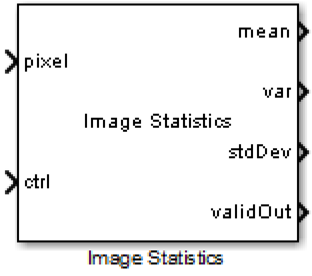
See Also

visionhdl.ROISelector | `Frame To Pixels`

Introduced in R2016a

Image Statistics

Mean, variance, and standard deviation



Library

visionhdlstatistics

Description

The Image Statistics block calculates the mean, variance, and standard deviation of streaming video data. Each calculation is performed over all pixels in the input region of interest (ROI). The block implements the calculations using hardware-efficient algorithms.

This block uses a streaming pixel interface with a bus for synchronization control signals. This interface enables the block to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox blocks. The block accepts a scalar pixel value and a bus containing five control signals. These signals indicate the validity of each pixel and the location of each pixel in the frame. To convert a pixel matrix into a pixel stream and these control signals, use the `Frame To Pixels` block. For a full description of the interface, see "Streaming Pixel Interface".

- To change the size and dimensions of the ROI, you can manipulate the input video stream control signals. See "Regions of Interest" on page 1-108.
- The number of valid pixels in the input image affects the accuracy of the mean approximation. To avoid approximation error, use an image that contains fewer than 64 pixels, a multiple of 64 pixels up to 64^2 pixels, a multiple of 64^2 pixels up

to 64^3 pixels, or a multiple of 64^3 pixels up to 64^4 pixels. For details of the mean approximation, see “Algorithm” on page 1-103.

- The block calculates statistics over frames up to 64^4 (16,777,216) pixels in size. This size supports HD frames.

Signal Attributes

Port	Direction	Description	Data Type
`pixel`	Input	Single image pixel specified as a scalar value.	• `uint8/uint16` • `fixdt(0,N,0)`, N = 8,9,...,16 `double` and `single` data types are supported for simulation but not for HDL code generation.
`ctrl`	Input	Control signals describing the validity of the pixel and the location of the pixel within the frame, specified as a bus containing five signals. See “Pixel Control Bus”.	`pixelcontrol`
`mean`	Output	Mean of the most recent input frame completed.	Same as `pixel`
`var`	Output	Variance of the most recent input frame completed.	Same as `pixel`
`stdDev`	Output	Standard deviation of the most recent input frame completed.	Same as `pixel`
`validOut`	Output	Computations completed. The block sets this output to `true` when the statistic outputs for a frame are ready.	`boolean`

Note: The block uses full-precision arithmetic for internal calculation. At the output, intermediate data is cast back to the input type using the following fixed-point settings: `RoundingMethod` = `Nearest`, and `OverflowAction` = `Saturate`. The table shows the output word length for each calculation, relative to the input word length (IWL).

Mean	Variance	Std. Deviation
IWL	2×IWL	2×IWL

Parameters

Enable mean output

Select this check box to calculate the mean of each input frame. If you clear this check box, the `mean` output does not show on the block.

Enable variance output

Select this check box to calculate the variance of each input frame. If you clear this check box, the `var` output does not show on the block.

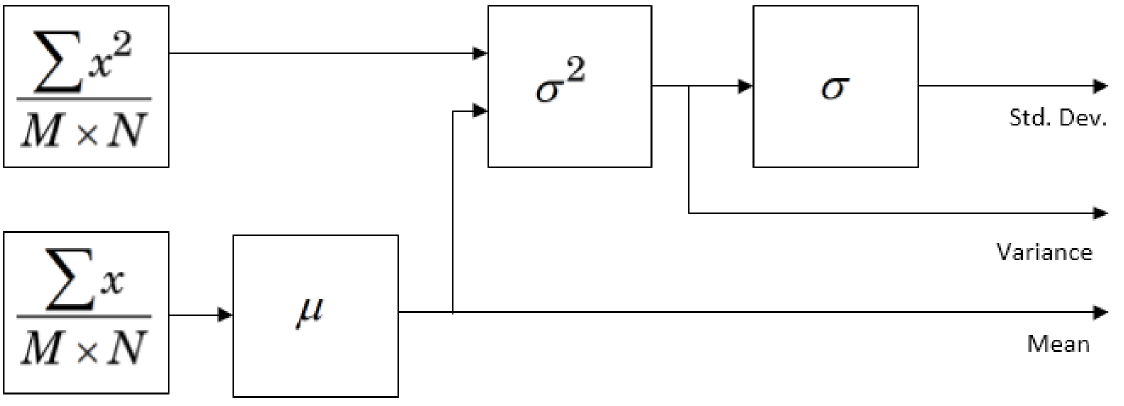
Enable std. deviation

Select this check box to calculate the standard deviation of each input frame. If you clear this check box, the `stdDev` output does not show on the block.

Algorithm

Architecture

The calculations of mean, variance, and standard deviation build off each other. For hardware efficiency, the calculation logic is shared as shown.



Mean

The equation to calculate the precise mean pixel value requires large internal word lengths and expensive division logic.

$$\mu = \frac{1}{M*N}\sum_{i=1}^{M}\sum_{j=1}^{N} x_{ij}$$

Instead of using this equation, the block calculates the mean by a series of four accumulators that compute the mean of a segment of pixels. First, find the sum of a window of 64 pixels, and normalize.

$$\mu_{L_1} = \frac{1}{64}\sum_{n=1}^{64} x_n$$

Then accumulate 64 of the previous windows, and normalize.

$$\mu_{L_2} = \frac{1}{64}\sum_{n=1}^{64} \mu_{nL_1}$$

A third accumulator sums 64 of the 64×64 windows, and normalizes the same way.

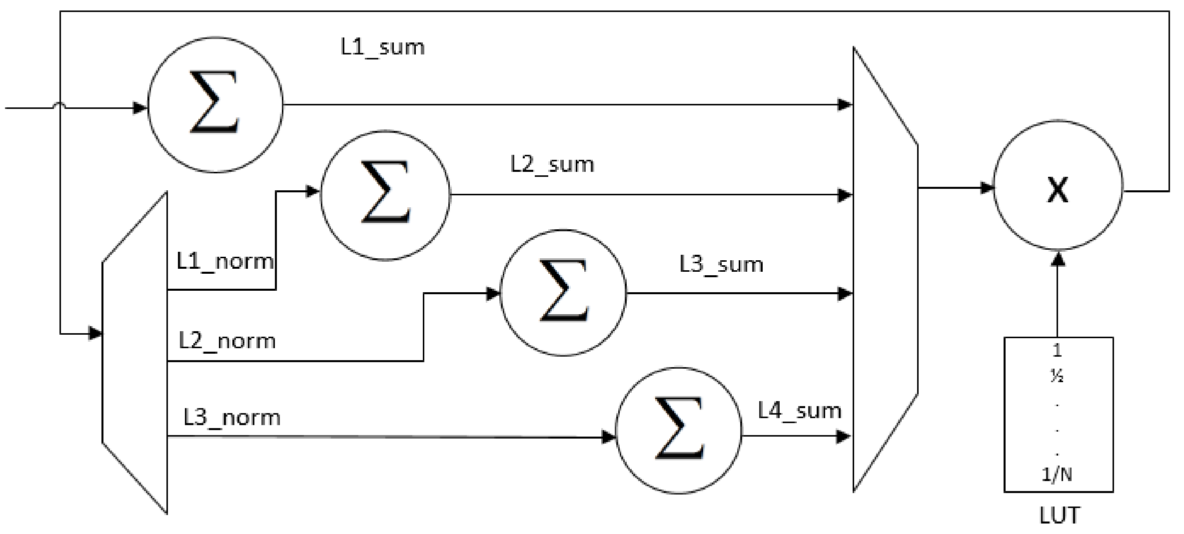
$$\mu_{L_3} = \frac{1}{64}\sum_{n=1}^{64} \mu_{nL_2}$$

The fourth accumulator sums 64 of the 64×64×64 windows and normalizes.

$$\mu_{L_4} = \frac{1}{64}\sum_{n=1}^{64} \mu_{nL_3}$$

Each valid pixel is accumulated as it arrives. Its location within a line or frame does not affect the accumulation logic.

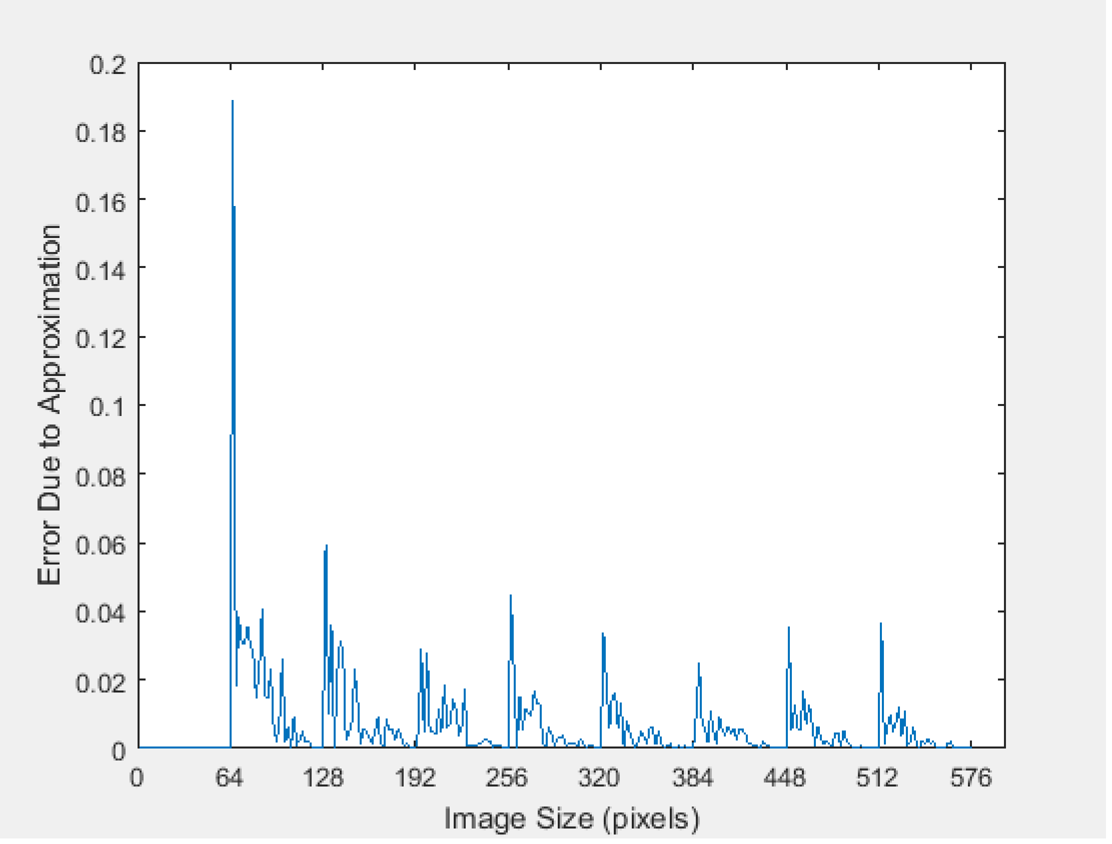
When `vEnd` is received, the block promotes any remaining data in the four levels of mean calculation to calculate the final output. If an accumulator counter is not at 64 when `vEnd` arrives, that level normalizes by the actual value of the counter. The constants for this multiplication are in a lookup table (LUT). The four accumulators share a single LUT and multiplier.



This method of mean calculation is accurate when the number of pixels in the frame aligns vEnd with the final accumulator rollover. This alignment occurs at level two when the frame contains a multiple of 64 pixels, and fewer than 64^2(4096) pixels. It occurs at level three when the frame contains a multiple of 4096 pixels. It occurs at level four when the frame contains a multiple of 64^3 pixels. This method is also accurate when the frame has fewer than 64 pixels, because only the first accumulator is needed.

However, when the number of pixels in the frame does not fit these conditions, the block must normalize the final accumulation before the counter reaches 64. This introduces an error in the normalization calculation at subsequent levels. The figure shows the normalization error introduced in the mean calculation by image sizes under 4096 pixels. The spikes occur where an image size is just over a multiple of 64 pixels.

For images larger than 4096 pixels, the same effect occurs at multiples of 4096 pixels, and at multiples of 64^3 pixels.



Variance

The block calculates variance of the input pixels using the following equation.

$$\sigma^2 = \left(\frac{1}{M * N}\sum_{i=1}^{M}\sum_{j=1}^{N} x_{ij}^2\right) - \mu^2$$

The mean and the mean of the squared input are calculated in parallel. The block calculates the mean of squares using the same approximation method used to calculate the mean, as described in the previous section.

Standard Deviation

The block calculates the square root of the variance using a pipelined bit-set-and-check algorithm. This algorithm computes the square root using addition and shifts rather than multipliers. For an N-bit input, the result has N bits of accuracy.

This method is hardware efficient for general inputs. If your data has known characteristics that allow for a more efficient square root implementation, you can disable the calculation in this block and construct your own logic from HDL-supported blocks.

Regions of Interest

Statistics are often calculated on small regions of interest (ROI) rather than an entire video frame. This block performs calculations on all valid pixels between `vStart` and `vEnd` signals in the `ctrl` bus, and does not track pixel location within the frame. You can manipulate the streaming control signals to reduce the size of a frame and delineate the boundaries of a region of interest before passing the video stream to this block. For an example that selects multiple small ROIs from a larger image, see "Multi-Zone Metering".

The Image Statistics block calculates statistics over frames up to 64^4 (16,777,216) pixels in size. If you provide an image with more than 64^4 pixels, the block calculates the requested statistics on only the first 16,777,216 pixels and then asserts `validOut`. The block ignores extra pixels until it receives a `vEnd` signal.

Latency

The latency from `vEnd` to `validOut` depends on the calculations you select.

When the block receives a `vEnd` signal that is `true`, it combines the remaining data in the four levels of mean calculation to calculate the final output. This final step takes 4 cycles per level, resulting in a maximum of 16 cycles of latency between the input `vEnd` signal and the `validOut` signal. Once the mean is available, the variance calculation takes 4 cycles. The square root logic requires input word length (IWL) cycles of latency.

If a calculation is not selected, and is not needed for other selected calculations, that logic is excluded from the generated HDL code.

The table shows the calculation logic and latency for various block configurations.

Mean	Variance	Std. Deviation	Logic Excluded From HDL	Latency (cycles)
✓	✓	✓		[4n]+4+IWL, (where n is the number of accumulator levels required for the input size)
✓			variance and square root	[4n]
	✓		square root	[4n]+4
		✓		[4n]+4+IWL
✓	✓		square root	[4n]+4
✓		✓		[4n]+4+IWL
	✓	✓		[4n]+4+IWL

Note: There must be at least 16 cycles between the `vEnd` signals on the input. This timing restriction enables the block to finish processing the current frame before the new one arrives.

If you are using a custom video format, set the horizontal blanking interval using the parameters of the Frame To Pixels block. The horizontal blanking interval is equal to **Total pixels per line** – **Active pixels per line**, or, equivalently, **Front porch** + **Back porch**. Standard streaming video formats use a horizontal blanking interval of about 25% of the frame width. This interval is much larger than the latency of the statistics operations.

See Also

visionhdl.ImageStatistics | `2-D Standard Deviation` | `2-D Mean` | `2-D Variance` | `Frame To Pixels`

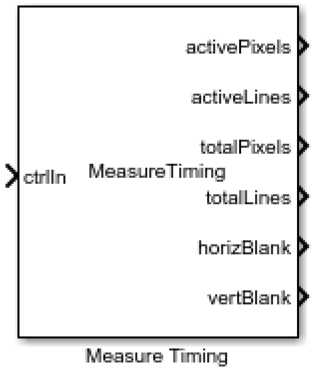
Related Examples

- "Multi-Zone Metering"

Introduced in R2015a

Measure Timing

Measure timing of pixel control bus input



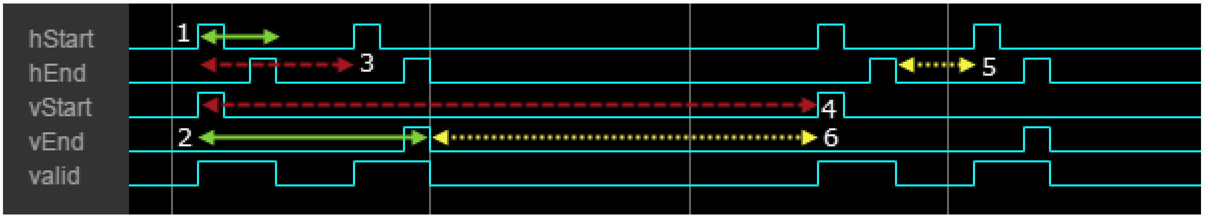
Library

visionhdlutilities

Description

The Measure Timing block measures the timing parameters of a video stream. The Vision HDL Toolbox streaming pixel protocol implements the timing of a video system, including inactive intervals between frames. These inactive intervals are called *blanking intervals*. Many Vision HDL Toolbox blocks require minimum blanking intervals. You can use the timing parameter measurements from this block to check that your video stream meets these requirements. If you manipulate the control signals of your video stream, you can use this block to verify the resulting signals.

To determine the parameters of each frame, the block measures the time steps between the control signals on the bus.



- 1 — Active pixels per line
- 2 — Active lines per frame (count `hStart` pulses)
- 3 — Total pixels per line
- 4 — Total lines per frame (cycles divided by total pixels per line)
- 5 — Horizontal blanking
- 6 — Vertical blanking (cycles, minus horizontal blanking, divided by total pixels per line)

For details on the pixel control bus and the dimensions of a video frame, see "Streaming Pixel Interface".

Note: Measurements from the first simulated frame are incorrect because some parameters require measurements between frames. Simulate at least two frames before using the results.

Signal Attributes

Port	Direction	Description	Data Type
`ctrl`	Input	Control signals describing the validity of the pixel and the location of the pixel within the frame, specified as a bus containing five signals. See "Pixel Control Bus".	`pixelcontrol`
`activePixels`	Output	Number of pixels in each line of	`uint32`

Port	Direction	Description	Data Type
		the active video frame. This value is measured between hStart and hEnd. See marker 1 in the diagram.	
activeLines	Output	Number of lines in the active video frame. This value is measured by counting hStart pulses between vStart and vEnd. See marker 2 in the diagram.	uint32
totalPixels	Output	Number of pixels in each line, including the horizontal blanking interval. This value is measured between hStart and the next hStart. See marker 3 in the diagram.	uint32
totalLines	Output	Number of lines in the frame, including the vertical blanking interval. This value is measured between vStart and the next vStart, divided by totalPixels. See marker 4 in the diagram.	uint32

Port	Direction	Description	Data Type
`horizBlank`	Output	Number of inactive pixels between lines of a frame. This value is measured between `hEnd` and the next `hStart`. See marker 5 in the diagram.	`uint32`
`vertBlank`	Output	Number of inactive lines between frames. This value is measured between `vEnd` and the next `vStart`, adjusted to remove `horizBlank`, and then divided by `totalPixels`. See marker 6 in the diagram.	`uint32`

See Also

visionhdl.MeasureTiming | `Frame To Pixels`

More About

- "Streaming Pixel Interface"

Introduced in R2016b

2

System Objects — Alphabetical List

visionhdl.ChromaResampler System object

Package: visionhdl

Downsample or upsample chrominance component

Description

`visionhdl.ChromaResampler` downsamples or upsamples a pixel stream.

- Downsampling reduces bandwidth and storage requirements in a video system by combining pixel chrominance components over multiple pixels. You can specify a filter to prevent aliasing, by selecting the default filter or by entering coefficients.
- Upsampling restores a signal to its original rate. You can use interpolation or replication to calculate the extra sample.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

The object accepts luma and the chrominance components. The object does not modify the luma component and applies delay to align with the resampled chrominance outputs. The rate of the output luma component is the same as the input.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object™, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`CR = visionhdl.ChromaResampler` returns a System object, `CR`, that downsamples from 4:4:4 to 4:2:2 and applies the default antialiasing filter.

CR = visionhdl.ChromaResampler(Name,Value) returns a chroma resampler System object, CR, with additional options specified by one or more Name,Value pair arguments. Name is a property name and Value is the corresponding value. Name must appear inside single quotes (''). You can specify several name-value pair arguments in any order as Name1,Value1,...,NameN,ValueN. Properties not specified retain their default values.

Properties

Resampling

Resampling format.

- 4:4:4 to 4:2:2 (default) — Perform a downsampling operation.
- 4:2:2 to 4:4:4 — Perform an upsampling operation.

AntialiasingFilterSource

Lowpass filter to accompany a downsample operation.

- Auto (default) — Built-in lowpass filter.
- Property — Filter using the coefficients in HorizontalFilterCoefficients property.
- None — No filtering of the input signal.

This property applies when you set Resampling to 4:4:4 to 4:2:2.

HorizontalFilterCoefficients

Coefficients for the antialiasing filter.

Enter the coefficients as a vector. This property applies when you set Resampling to 4:4:4 to 4:2:2 and Antialiasing filter to Property.

Default: [0.2,0.6,0.2]

InterpolationFilter

Interpolation method for an upsample operation.

- `Linear` (default) — Linear interpolation to calculate the missing values.
- `Pixel replication` — Repeat the chrominance value of the preceding pixel to create the missing pixel.

This property applies when you set `Resampling` to `4:2:2 to 4:4:4`.

RoundingMethod

Rounding mode used for fixed-point operations.

The object uses fixed-point arithmetic for internal calculations when the input is any integer or fixed-point data type. This option does not apply when the input data type is `single` or `double`.

Default: `Floor`

OverflowAction

Overflow action used for fixed-point operations.

The object uses fixed-point arithmetic for internal calculations when the input is any integer or fixed-point data type. This option does not apply when the input data type is `single` or `double`.

Default: `Wrap`

CustomCoefficientsDataType

Data type for the antialiasing filter coefficients.

Specify a custom data type as a character vector. This parameter applies when you set `Antialiasing filter` to `Property` or `Auto`.

Default: `'fixdt(1,16,0)'`

Methods

clone	Create object with the same property values

isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics
step	Compute next pixel in upsampled or downsampled pixel stream

Examples

Downsample a Y'CbCr Image

Resample a 4:4:4 Y'CbCr image to 4:2:2. The example also shows how to convert a R'G'B' input image to Y'CbCr color space.

Prepare a test image by selecting a portion of an image file.

```
frmActivePixels = 64;
frmActiveLines = 48;
frmOrig = imread('fabric.png');
frmInput = frmOrig(1:frmActiveLines,1:frmActivePixels,:);
```

Create a serializer and specify the size of inactive pixel regions. The number of padding pixels on each line must be greater than the latency of each pixel-processing object.

```
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',3,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines,...
      'TotalPixelsPerLine',frmActivePixels+40,...
      'TotalVideoLines',frmActiveLines+10,...
      'StartingActiveLine',6,...
      'FrontPorch',5);
```

Create a color space converter and resampler, using the default property values. The default conversion is 'RGB to YCbCr'. The default resampling mode is '4:4:4 to 4:2:2'. The default anti-aliasing filter is a 29-tap lowpass filter. This gives the object a latency of 30 cycles.

```
convert2ycbcr = visionhdl.ColorSpaceConverter();
downsampler = visionhdl.ChromaResampler();
```

Serialize the test image using the serializer object. `pixIn` is a `numPixelsPerFrame`-by-3 matrix. `ctrlIn` is a vector of control signal structures. Preallocate vectors for the output signals.

```
[pixIn,ctrlIn] = step(frm2pix,frmInput);

[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
pix444 = zeros(numPixelsPerFrame,3,'uint8');
ctrl444  = repmat(pixelcontrolstruct,numPixelsPerFrame,1);
pix422 = zeros(numPixelsPerFrame,3,'uint8');
ctrl422 = repmat(pixelcontrolstruct,numPixelsPerFrame,1);
```

For each pixel in the stream, convert to YCbCr, then downsample.

```
for p = 1:numPixelsPerFrame
    [pix444(p,:),ctrl444(p)] = step(convert2ycbcr,pixIn(p,:),ctrlIn(p));
    [pix422(p,:),ctrl422(p)] = step(downsampler,pix444(p,:),ctrl444(p));
end
```

Create deserializers with a format matching that of the serializer. Convert the 4:4:4 and 4:2:2 pixel streams back to image frames.

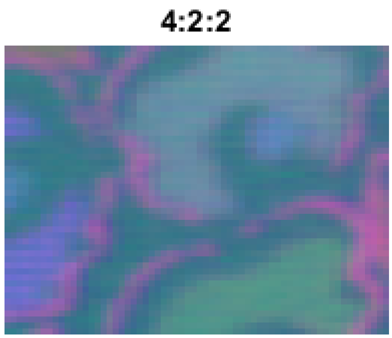
```
pix2frm444 = visionhdl.PixelsToFrame(...
      'NumComponents',3,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines);

pix2frm422 = visionhdl.PixelsToFrame(...
     'NumComponents',3,...
     'VideoFormat','custom',...
     'ActivePixelsPerLine',frmActivePixels,...
     'ActiveVideoLines',frmActiveLines);

[frm444,frmValid] = step(pix2frm444,pix444,ctrl444);
[frm422,frmValid] = step(pix2frm422,pix422,ctrl422);
```

There are the same number of pixels in the 4:2:2 and 4:4:4 pixel-streams and frames. To examine the resampled data, regroup the pixel data for the first 8 pixels of the first line. The first row is the Y elements of the pixels, the second row is the Cb elements, and the third row is the Cr elements. Notice that, in the 4:2:2 data, the Cb and Cr elements change only every second sample.

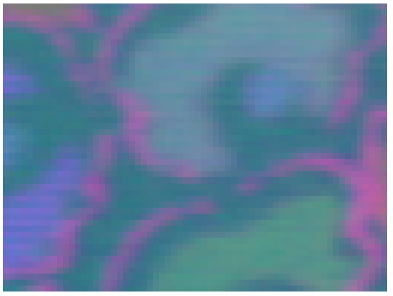
```
YCbCr444 = [frm444(1,1:8,1); frm444(1,1:8,2); frm444(1,1:8,3)]
YCbCr422 = [frm422(1,1:8,1); frm422(1,1:8,2); frm422(1,1:8,3)]
```

```
figure
imshow(frm422,'InitialMagnification',300)
title '4:2:2'
figure
imshow(frm444,'InitialMagnification',300)
title '4:4:4'
```

```
YCbCr444 =

  3×8 uint8 matrix

   132   134   129   124   125   122   118   119
   116   118   119   122   122   121   123   123
   135   131   125   121   119   116   118   118


YCbCr422 =

  3×8 uint8 matrix

   132   134   129   124   125   122   118   119
   116   116   120   120   122   122   123   123
   135   135   126   126   119   119   118   118
```



4:4:4



Algorithm

This object implements the algorithms described on the `Chroma Resampler` block reference page.

See Also

`Chroma Resampler` | visionhdl.FrameToPixels | vision.ChromaResampler

Introduced in R2015a

clone

System object: visionhdl.ChromaResampler
Package: visionhdl

Create object with the same property values

Syntax

`newCR = clone(CR)`

Description

`newCR = clone(CR)` creates another instance of the `ChromaResampler` System object, `CR`, that has the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

`CR`

`visionhdl.ChromaResampler` System object

Output Arguments

`newCR`

New `ChromaResampler` System object that has the same property values as the original System object.

Introduced in R2015a

isLocked

System object: visionhdl.ChromaResampler
Package: visionhdl

Locked status (logical)

Syntax

`TF = isLocked(CR)`

Description

`TF = isLocked(CR)` returns the locked status, `TF`, of the `ChromaResampler` System object, `CR`.

Introduced in R2015a

release

System object: visionhdl.ChromaResampler
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

```
release(CR)
```

Description

`release(CR)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

`CR`

`visionhdl.ChromaResampler` System object

Introduced in R2015a

step

System object: visionhdl.ChromaResampler
Package: visionhdl

Compute next pixel in upsampled or downsampled pixel stream

Syntax

`[pixelOut,ctrlOut] = step(CR,pixelIn,ctrlIn)`

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[pixelOut,ctrlOut] = step(CR,pixelIn,ctrlIn)` computes the next output pixel, `pixelOut`, in the resampled video stream. The pixel data arguments, `pixelIn` and `pixelOut`, are vectors of three values representing a pixel in Y'CbCr color space. The luma component and control signals, `ctrlIn`, are passed through and aligned with the output pixel stream.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

CR

`visionhdl.ChromaResampler` System object.

pixelIn

Single pixel in gamma-corrected Y'CbCr color space, specified as a vector of three values.

Supported data types:

- `uint8` or `uint16`
- `fixdt(0,N,0)`, N = 8,9,...,16
- `double` and `single` data types are supported for simulation but not for HDL code generation.

ctrlIn

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five `logical` signals. See "Pixel Control Structure".

Output Arguments

pixelOut

Single pixel in gamma-corrected Y'CbCr color space, returned as a vector of three values.

Supported data types:

- `uint8` or `uint16`
- `fixdt(0,N,0)`, N = 8,9,...,16
- `double` and `single` data types are supported for simulation but not for HDL code generation.

ctrlOut

Control signals indicating the validity of the output pixel and the location of the pixel within the frame, returned as a structure containing five `logical` signals. See “Pixel Control Structure”.

Introduced in R2015a

visionhdl.ColorSpaceConverter System object

Package: visionhdl

Convert color information between color spaces

Description

`visionhdl.ColorSpaceConverter` converts between R'G'B' and Y'CbCr color spaces, and also converts R'G'B' to intensity.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: The `ColorSpaceConverter` System object operates on gamma-corrected color spaces. However, to simplify use of the System object, the property arguments do not include the prime notation.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`CSC = visionhdl.ColorSpaceConverter` returns a System object, `CSC`, that converts R'G'B' to Y'CbCr using the Rec. 601 (SDTV) standard.

`CSC = visionhdl.ColorSpaceConverter(Name,Value)` returns a System object, `CSC`, with additional options specified by one or more `Name`,`Value` pair arguments. `Name` is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as Name1,Value1,...,NameN,ValueN. Properties not specified retain their default values.

Properties

Conversion

Conversion that the object performs on the input video stream.

- RGB to YCbCr (default)
- YCbCr to RGB
- RGB to intensity

The step method accepts input as a vector of three values representing a single pixel. If you choose RGB to intensity, the output is a scalar value. Otherwise, the output is a vector of three values.

ConversionStandard

Conversion equation to use on the input video stream.

- Rec. 601 (SDTV) (default)
- Rec. 709 (HDTV)

This property does not apply when you set Conversion to RGB to intensity.

ScanningStandard

Scanning standard to use for HDTV conversion.

- 1250/50/2:1 (default)
- 1125/60/2:1

This property applies when you set ConversionStandard to Rec. 709 (HDTV).

Methods

clone	Create object having the same property values

isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics
step	Convert one pixel between color spaces

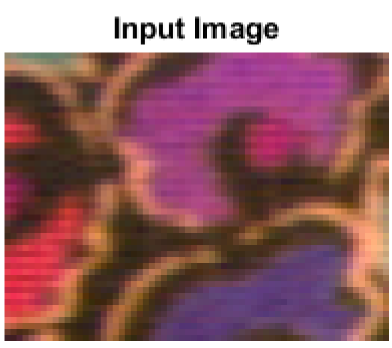
Examples

Convert a Color Image to Grayscale

This example shows how to convert pixel stream data to a different color space.

Set the dimensions of the test image and load a color source image. Select a portion of the image matching the desired test size.

```
frmActivePixels = 64;
frmActiveLines = 48;
frmOrig = imread('fabric.png');
frmInput = frmOrig(1:frmActiveLines,1:frmActivePixels,:);
figure
imshow(frmInput,'InitialMagnification',300)
title 'Input Image'
```



Create a serializer object and specify size of inactive pixel regions.

```
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',3,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines,...
      'TotalPixelsPerLine',frmActivePixels+10,...
      'TotalVideoLines',frmActiveLines+10,...
      'StartingActiveLine',6,...
      'FrontPorch',5);
```

Create a color space converter object. Select a conversion from RGB to grayscale.

```
convertrgb2gray = visionhdl.ColorSpaceConverter(...
      'Conversion','RGB to intensity');
```

Serialize the test image. `pixIn` is a numPixelsPerFrame-by-3 matrix. `ctrlIn` is a vector of control signal structures.

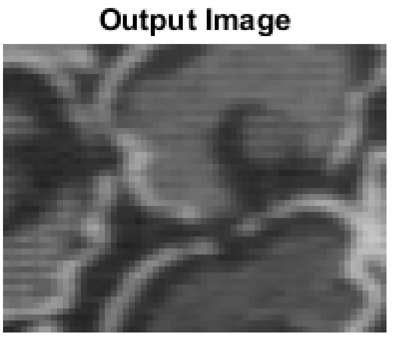
```
[pixIn,ctrlIn] = step(frm2pix,frmInput);
```

Set up variables, and convert each pixel in the stream to the new color space.

```
[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
pixOut = zeros(numPixelsPerFrame,1,'uint8');
ctrlOut = repmat(pixelcontrolstruct,numPixelsPerFrame,1);
for p = 1:numPixelsPerFrame
    [pixOut(p),ctrlOut(p)] = step(convertrgb2gray,pixIn(p,:),ctrlIn(p));
end
```

Create a deserializer object with format matching that of the serializer. Convert the pixel stream to an image frame, and display the grayscale output image.

```
pix2frm = visionhdl.PixelsToFrame(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines);
[frmOutput,frmValid] = step(pix2frm,pixOut,ctrlOut);
if frmValid
    figure
    imshow(frmOutput,'InitialMagnification',300)
    title 'Output Image'
end
```



Algorithm

This object implements the algorithms described on the `Color Space Converter` block reference page.

See Also

`Colorspace Converter` | vision.ColorSpaceConverter | `rgb2ycbcr` | visionhdl.FrameToPixels | `ycbcr2rgb` | `rgb2gray`

Introduced in R2015a

clone

System object: visionhdl.ColorSpaceConverter
Package: visionhdl

Create object having the same property values

Syntax

`newCSC = clone(CSC)`

Description

`newCSC = clone(CSC)` creates another instance of the `ColorSpaceConverter` System object, `CSC`, that has the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

`CSC`

`visionhdl.ColorSpaceConverter` System object

Output Arguments

`newCSC`

New `ColorSpaceConverter` System object that has the same property values as the original System object.

Introduced in R2015a

isLocked

System object: visionhdl.ColorSpaceConverter
Package: visionhdl

Locked status (logical)

Syntax

```
TF = isLocked(CSC)
```

Description

`TF = isLocked(CSC)` returns the locked status, `TF`, of the `ColorSpaceConverter` System object, `CSC`.

Introduced in R2015a

release

System object: visionhdl.ColorSpaceConverter
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

`release(CSC)`

Description

`release(CSC)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

`CSC`

`visionhdl.ColorSpaceConverter` System object

Introduced in R2015a

step

System object: visionhdl.ColorSpaceConverter
Package: visionhdl

Convert one pixel between color spaces

Syntax

`[pixelOut,ctrlOut] = step(CSC,pixelIn,ctrlIn)`

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[pixelOut,ctrlOut] = step(CSC,pixelIn,ctrlIn)` converts a single pixel from one color space to another. The input, `pixelIn` is a vector of three values representing one pixel in R'G'B' or Y'CbCr color space. If the `Conversion` property is set to `RGB to YCbCr` or `YCbCr to RGB`, then `pixelOut` is a vector of three values representing one pixel. If the `Conversion` property is set to `RGB to intensity`, then `pixelOut` is a scalar value representing one pixel.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: The `ColorSpaceConverter` System object operates on gamma-corrected color spaces. However, to simplify use of the System object, the property arguments do not include the prime notation.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

csc

`visionhdl.ColorSpaceConverter` System object.

pixelIn

Input pixel in gamma-corrected R'G'B' or Y'CbCr color space, specified as a vector of unsigned integer values.

Supported data types:

- `uint8` or `uint16`
- `fixdt(0,N,0)`, N = 8,9,...,16
- `double` and `single` data types are supported for simulation but not for HDL code generation.

ctrlIn

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five `logical` signals. See “Pixel Control Structure”.

Output Arguments

pixelOut

Output pixel specified as a vector of three unsigned integer values, or a scalar unsigned integer value.

- If you set the `Conversion` property to `RGB to YCbCr` or `YCbCr to RGB`, then `pixelOut` is a vector representing the pixel in gamma-corrected color space.

- If you set the `Conversion` property to `RGB to intensity`, then `pixelOut` is a scalar representing pixel intensity.

Supported data types:

- `uint8` or `uint16`
- `fixdt(0,N,0)`, N = 8,9,....,16
- `double` and `single` data types are supported for simulation but not for HDL code generation.

`ctrlOut`

Control signals indicating the validity of the output pixel and the location of the pixel within the frame, returned as a structure containing five `logical` signals. See “Pixel Control Structure”.

Introduced in R2015a

visionhdl.Closing System object

Package: visionhdl

Morphological closing of binary pixel data

Description

`visionhdl.Closing` performs morphological dilation, followed by morphological erosion, using the same neighborhood for both calculations. The object operates on a stream of binary intensity values.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`C = visionhdl.Closing` returns a System object, `C`, that performs morphological closing on a binary pixel stream.

`C = visionhdl.Closing(Name,Value)` returns a System object, `C`, with additional options specified by one or more `Name,Value` pair arguments. `Name` is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Properties not specified retain their default values.

Properties

`Neighborhood`

Pixel neighborhood, specified as a matrix or vector of ones and zeros.

The object supports neighborhoods of up to 32×32 pixels. To use a structuring element, specify `Neighborhood` as `getnhood(strel(`*`shape`*`))`.

Default: `[0,1,0;1,1,1;0,1,0]`

`LineBufferSize`

Size of the line memory buffer, specified as a scalar integer.

Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the object uses the next largest power of two. The object allocates (*neighborhood lines* – 1)-by-`LineBufferSize` memory locations to store the pixels.

Default: `2048`

Methods

clone	Create object having the same property values
isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics
step	Report closed pixel value based on neighborhood

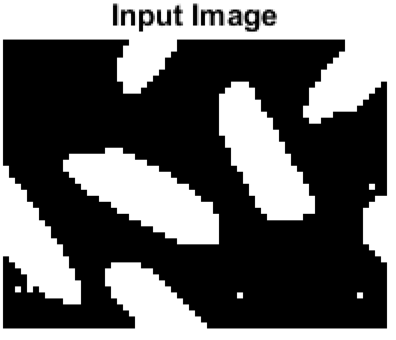
Examples

Morphological Close

Perform morphological close on a thumbnail image.

Load a source image from a file. Select a portion of the image that matches the desired test size. This source image contains `uint8` pixel intensity values. Apply a threshold to convert to binary pixel data.

```
frmOrig = imread('rice.png');
frmActivePixels = 64;
frmActiveLines = 48;
frmInput = frmOrig(1:frmActiveLines,1:frmActivePixels);
frmInput = frmInput>128;
figure
imshow(frmInput,'InitialMagnification',300)
title 'Input Image'
```



Create a serializer object and define inactive pixel regions. Make the number of inactive pixels following each active line at least double the horizontal size of the neighborhood. Make the number of lines following each frame at least double the vertical size of the neighborhood.

```
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines,...
      'TotalPixelsPerLine',frmActivePixels+20,...
      'TotalVideoLines',frmActiveLines+10,...
      'StartingActiveLine',3,...
```

```
      'FrontPorch',10);
```

Create a filter object.

```
 mclose = visionhdl.Closing( ...
          'Neighborhood',getnhood(strel('disk',4)));
```

Serialize the test image by calling `step` on the serializer object. `pixIn` is a vector of intensity values. `ctrlIn` is a vector of control signal structures.

```
[pixIn,ctrlIn] = step(frm2pix,frmInput);
```

Prepare to process pixels by preallocating output vectors.

```
[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
pixOut = false(numPixelsPerFrame,1);
ctrlOut  = repmat(pixelcontrolstruct,numPixelsPerFrame,1);
```

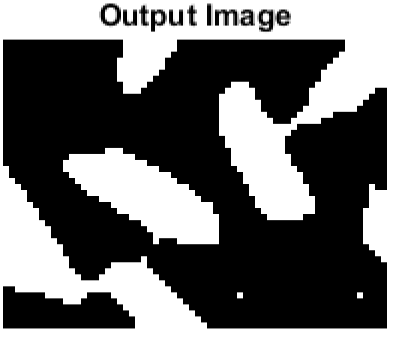
For each pixel in the padded frame, compute the morphed value. Monitor the control signals to determine latency of the object. The latency of a configuration depends on the number of active pixels in a line and the size of the neighborhood

```
foundValIn = false;
foundValOut = false;
for p = 1:numPixelsPerFrame
    if (ctrlIn(p).valid && foundValIn==0)
        foundValIn = p;
    end
    [pixOut(p),ctrlOut(p)] = step(mclose,pixIn(p),ctrlIn(p));
    if (ctrlOut(p).valid && foundValOut==0)
        foundValOut = p;
    end
end
sprintf('object latency is %d cycles',foundValOut-foundValIn)
```

```
ans =

object latency is 546 cycles
```

Create a deserializer object with a format matching that of the serializer. Convert the pixel stream to an image frame by calling `step` on the deserializer object. Display the resulting image.

```
pix2frm = visionhdl.PixelsToFrame(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines);
[frmOutput,frmValid] = step(pix2frm,pixOut,ctrlOut);
if frmValid
    figure
    imshow(frmOutput, 'InitialMagnification',300)
    title 'Output Image'
end
```



Algorithm

This object implements the algorithms described on the `Closing` block reference page.

See Also

visionhdl.Opening | vision.MorphologicalClose | visionhdl.FrameToPixels | visionhdl.Erosion | visionhdl.Dilation | `Closing` | `imclose`

Introduced in R2015a

clone

System object: visionhdl.Closing
Package: visionhdl

Create object having the same property values

Syntax

`newC = clone(C)`

Description

`newC = clone(C)` creates another instance of the `Closing` System object, `C`, that has the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

`C`

`visionhdl.Closing` System object

Output Arguments

`newC`

New `Closing` System object that has the same property values as the original System object.

Introduced in R2015a

isLocked

System object: visionhdl.Closing
Package: visionhdl

Locked status (logical)

Syntax

```
TF = isLocked(C)
```

Description

`TF = isLocked(C)` returns the locked status, `TF`, of the `Closing` System object, `C`.

Introduced in R2015a

release

System object: visionhdl.Closing
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

```
release(C)
```

Description

`release(C)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

`C`

`visionhdl.Closing` System object

Introduced in R2015a

step

System object: visionhdl.Closing
Package: visionhdl

Report closed pixel value based on neighborhood

Syntax

`[pixelOut,ctrlOut] = step(C,pixelIn,ctrlIn)`

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[pixelOut,ctrlOut] = step(C,pixelIn,ctrlIn)` returns the next binary pixel value, `pixelOut`, resulting from a morphological close operation on the neighborhood around each input binary pixel, `pixelIn`.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

C

`visionhdl.Closing` System object

pixelIn

Single pixel, specified as a scalar `logical` value.

ctrlIn

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five `logical` signals. See “Pixel Control Structure”.

Output Arguments

pixelOut

Single pixel transformed by a morphological operation, returned as a scalar `logical` value.

ctrlOut

Control signals indicating the validity of the output pixel and the location of the pixel within the frame, returned as a structure containing five `logical` signals. See “Pixel Control Structure”.

Introduced in R2015a

visionhdl.GrayscaleClosing System object

Package: visionhdl

Morphological closing of grayscale pixel data

Description

`visionhdl.GrayscaleClosing` performs a morphological dilation operation, followed by a morphological erosion operation, using the same neighborhood for both calculations. The object operates on a stream of pixel intensity values. You can specify a neighborhood, or structuring element, of up to 32×32 pixels. For line, square, or rectangle structuring elements more than 8 pixels wide, the object uses the Van Herk algorithm to find the maximum and minimum. For structuring elements less than 8 pixels wide, or that contain zero elements, the object implements a pipelined comparison tree to find the maximum and minimum.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`C = visionhdl.GrayscaleClosing` returns a System object, `C`, that performs morphological closing on a pixel stream.

`C = visionhdl.GrayscaleClosing(Name,Value)` returns a System object, `C`, with additional options specified by one or more `Name`,`Value` pair arguments. `Name`

is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Properties not specified retain their default values.

Properties

`Neighborhood`

Pixel neighborhood, specified as a matrix or vector of ones and zeros.

The object supports neighborhoods of up to 32×32 pixels. To use a structuring element, specify `Neighborhood` as `getnhood(strel(`*`shape`*`))`. The minimum neighborhood size is a 2×2 matrix, or a 2×1 column vector. If the neighborhood is a row vector, it must be at least 8 columns wide and contain no zeros.

Default: `ones(3,3)`

`LineBufferSize`

Size of the line memory buffer, specified as a scalar integer.

Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the object uses the next largest power of two. The object allocates (*neighborhood lines* – 1)-by-`LineBufferSize` memory locations to store the pixels.

Default: `2048`

Methods

clone	Create object having the same property values
isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics

step	Report closed pixel value based on neighborhood

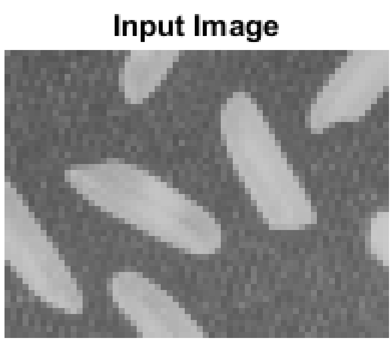
Examples

Grayscale Morphological Closing

Perform morphological closing on a grayscale thumbnail image.

Load a source image from a file. Select a portion of the image matching the desired test size.

```
frmOrig = imread('rice.png');
frmActivePixels = 64;
frmActiveLines = 48;
frmInput = frmOrig(1:frmActiveLines,1:frmActivePixels);
imshow(frmInput,'InitialMagnification',300)
title 'Input Image'
```



Create a serializer object and define the inactive pixel regions. Make the number of inactive pixels following each active line at least double the horizontal size of the neighborhood. Make the number of lines following each frame at least double the vertical size of the neighborhood.

```
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines,...
      'TotalPixelsPerLine',frmActivePixels+20,...
      'TotalVideoLines',frmActiveLines+20,...
      'StartingActiveLine',3,...
      'FrontPorch',10);
```

Create a filter object.

```
mclose = visionhdl.GrayscaleClosing( ...
          'Neighborhood',ones(5,5));
```

Serialize the test image by calling `step` on the serializer object. `pixIn` is a vector of intensity values. `ctrlIn` is a vector of control signal structures.

```
[pixIn,ctrlIn] = step(frm2pix,frmInput);
```

Prepare to process pixels by preallocating output vectors.

```
[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
pixOut = uint8(zeros(numPixelsPerFrame,1));
ctrlOut = repmat(pixelcontrolstruct,numPixelsPerFrame,1);
```

For each pixel in the padded frame, compute the morphed value. Monitor the control signals to determine the latency of the object. The latency of a configuration depends on the number of active pixels in a line and the size of the neighborhood.

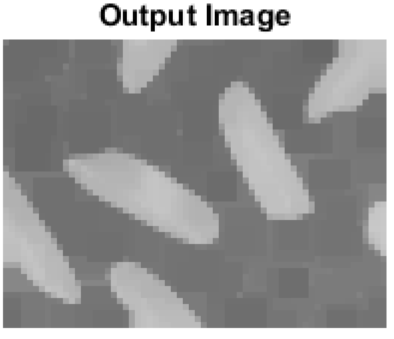
```
foundValIn = false;
foundValOut = false;
for p = 1:numPixelsPerFrame
    if (ctrlIn(p).valid && foundValIn==0)
        foundValIn = p;
    end
    [pixOut(p),ctrlOut(p)] = step(mclose,pixIn(p),ctrlIn(p));
    if (ctrlOut(p).valid && foundValOut==0)
        foundValOut = p;
    end
end
sprintf('object latency is %d cycles',foundValOut-foundValIn)
```

```
ans =
```

```
object latency is 388 cycles
```

Create a deserializer object with a format matching that of the serializer. Convert the pixel stream to an image frame by calling `step` on the deserializer object. Display the resulting image.

```
pix2frm = visionhdl.PixelsToFrame(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines);
[frmOutput,frmValid] = step(pix2frm,pixOut,ctrlOut);
if frmValid
    figure
    imshow(frmOutput, 'InitialMagnification',300)
    title 'Output Image'
end
```



Algorithm

This object implements the algorithms described on the `Grayscale Closing` block reference page.

See Also

visionhdl.GrayscaleDilation | visionhdl.GrayscaleOpening | `Grayscale Closing` | vision.MorphologicalClose | `imclose` | visionhdl.GrayscaleErosion | visionhdl.FrameToPixels

Introduced in R2016a

clone

System object: visionhdl.GrayscaleClosing
Package: visionhdl

Create object having the same property values

Syntax

```
newC = clone(C)
```

Description

`newC = clone(C)` creates another instance of the `GrayscaleClosing` System object, `C`, that has the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

`C`

`visionhdl.GrayscaleClosing` System object

Output Arguments

`newC`

New `GrayscaleClosing` System object that has the same property values as the original System object.

Introduced in R2016a

isLocked

System object: visionhdl.GrayscaleClosing
Package: visionhdl

Locked status (logical)

Syntax

```
TF = isLocked(C)
```

Description

`TF = isLocked(C)` returns the locked status, `TF`, of the `GrayscaleClosing` System object, `C`.

Introduced in R2016a

release

System object: visionhdl.GrayscaleClosing
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

```
release(C)
```

Description

`release(C)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

`C`

`visionhdl.GrayscaleClosing` System object

Introduced in R2016a

step

System object: visionhdl.GrayscaleClosing
Package: visionhdl

Report closed pixel value based on neighborhood

Syntax

```
[pixelOut,ctrlOut] = step(C,pixelIn,ctrlIn)
```

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[pixelOut,ctrlOut] = step(C,pixelIn,ctrlIn)` returns the next pixel value, `pixelOut`, resulting from morphological closing on the neighborhood around each input pixel intensity value, `pixelIn`.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

C

`visionhdl.Closing` System object

pixelIn

Single pixel, specified as a scalar value.

Supported data types:

- `uint8`, `uint16`,`uint32`
- `fixdt(0,N,M)`
- `double` and `single` data types are supported for simulation but not for HDL code generation.

ctrlIn

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five `logical` signals. See “Pixel Control Structure”.

Output Arguments

pixelOut

Single pixel transformed by a morphological operation, returned as a scalar value.

The data type is the same as the data type of `pixelIn`.

ctrlOut

Control signals indicating the validity of the output pixel and the location of the pixel within the frame, returned as a structure containing five `logical` signals. See “Pixel Control Structure”.

Introduced in R2016a

visionhdl.DemosaicInterpolator System object

Package: visionhdl

Construct full RGB pixel data from Bayer pattern pixels

Description

`visionhdl.DemosaicInterpolator` provides a Bayer pattern interpolation filter for streaming video data. You can select a low complexity bilinear interpolation, or a moderate complexity gradient-corrected bilinear interpolation. The object implements the calculations using hardware-efficient algorithms for HDL code generation.

- The object performs bilinear interpolation on a 3×3 pixel window using only additions and bit shifts.
- The object performs gradient correction on a 5×5 pixel window. The object implements the calculation using bit shift, addition, and low order Canonical Signed Digit (CSD) multiply.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`D = visionhdl.DemosaicInterpolator` returns a System object, `D`, that interpolates R'G'B' data from a Bayer pattern pixel stream.

`D = visionhdl.DemosaicInterpolator(Name,Value)` returns a System object, `D`, with additional options specified by one or more `Name`,`Value` pair arguments. `Name` is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Properties not specified retain their default values.

Properties

`InterpolationAlgorithm`

Algorithm the object uses to calculate the missing pixel values.

- `Bilinear` — Average of the pixel values in the surrounding 3×3 neighborhood.
- `Gradient-corrected linear` (default) — Bilinear average, corrected for intensity gradient.

`SensorAlignment`

Color sequence of the pixels in the input stream.

Specify the sequence of R, G, and B pixels that correspond to the 2-by-2 block of pixels in the top-left corner of the input image. Specify the sequence in left-to-right, top-to-bottom order. For instance, the default value, `RGGB`, represents an image with this pattern.

R	G
G	B

`LineBufferSize`

Size of the line memory buffer, specified as a scalar integer.

Choose a power of 2 that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the object uses the next largest power of two. When you set `InterpolationAlgorithm` to `Bilinear`, the object allocates 2-by-`LineBufferSize` memory locations. When you set `InterpolationAlgorithm` to `Gradient-corrected linear`, the object allocates 4-by-`LineBufferSize` memory locations.

Default: 2048

Methods

clone	Create object having the same property values

isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics
step	Demosaic a Bayer pattern video stream

Examples

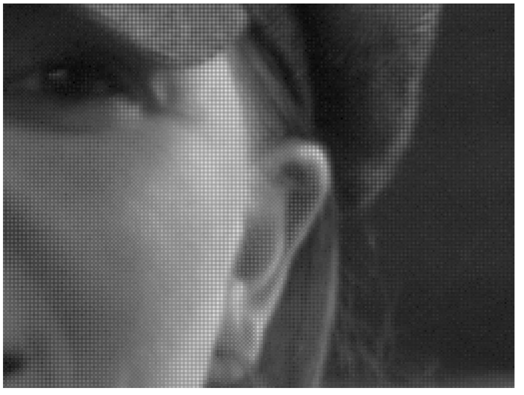
Demosaic a Bayer Pattern Image

This example constructs full RGB pixel data from a Bayer pattern thumbnail image.

Set the dimensions of the test image. Load the source image file. This image is in Bayer pattern: each pixel is represented by one value, alternating green values with red and blue values. Then select a portion of the image matching the desired test size. These offsets select the face of the woman in the image.

```
frmActivePixels = 256;
frmActiveLines = 192;
frmOrig = imread('mandi.tif');
frmInput = frmOrig(900:899+frmActiveLines, 2350:2349+frmActivePixels);
figure
imshow(frmInput)
title 'Input Image'
```

Input Image



Create a serializer object and specify size of the inactive pixel regions.

```
frm2pix = visionhdl.FrameToPixels(...
       'NumComponents',1,...
       'VideoFormat','custom',...
       'ActivePixelsPerLine',frmActivePixels,...
       'ActiveVideoLines',frmActiveLines,...
       'TotalPixelsPerLine',frmActivePixels+10,...
       'TotalVideoLines',frmActiveLines+10,...
       'StartingActiveLine',6,...
       'FrontPorch',5);
```

Create an interpolator object. Specify the sequence of color values matching the 2-by-2 pixels in the top-left corner of the image.

```
BayerInterpolator = visionhdl.DemosaicInterpolator(...
    'SensorAlignment', 'RGGB');
```

Serialize the test image. `pixIn` is a vector of pixel values. `ctrlIn` is a vector of control signal structures.

```
[pixIn,ctrlIn] = step(frm2pix,frmInput);
```

Set up variables, and generate the {R,G,B} triplet for each pixel in the stream. This example prints a progress message every 32 lines.

```
[pixels,lines,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
ctrlOut  = repmat(pixelcontrolstruct,numPixelsPerFrame,1);
pixOut = zeros(numPixelsPerFrame,3,'uint8');
lineCount = 1;
for p = 1:numPixelsPerFrame
    if ctrlIn(p).hEnd
        lineCount = lineCount+1;
        if mod(lineCount,32)==0
            fprintf('Processing... line %d\n',lineCount);
        end
    end
    [pixOut(p,:),ctrlOut(p)] = step(BayerInterpolator,pixIn(p),ctrlIn(p));
end
```

```
Processing... line 32
Processing... line 64
Processing... line 96
Processing... line 128
Processing... line 160
Processing... line 192
```

Create a deserializer object with a format matching that of the serializer. Convert the pixel stream to an image frame, and display the result.

```
pix2frm = visionhdl.PixelsToFrame(...
      'NumComponents',3,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines);
[frmOutput,frmValid] = step(pix2frm,pixOut,ctrlOut);
if frmValid
    figure
    imshow(frmOutput)
    title 'Output Image'
end
```

Output Image

Algorithm

This object implements the algorithms described on the `Demosaic Interpolator` block reference page.

See Also

`Demosaic Interpolator` | vision.DemosaicInterpolator | `demosaic` | visionhdl.FrameToPixels

Introduced in R2015a

clone

System object: visionhdl.DemosaicInterpolator
Package: visionhdl

Create object having the same property values

Syntax

`newD = clone(D)`

Description

`newD = clone(D)` creates another instance of the `DemosaicInterpolator` System object, `D`, that has the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

`D`

`visionhdl.DemosaicInterpolator` System object

Output Arguments

`newD`

New `DemosaicInterpolator` System object that has the same property values as the original System object.

Introduced in R2015a

isLocked

System object: visionhdl.DemosaicInterpolator
Package: visionhdl

Locked status (logical)

Syntax

```
TF = isLocked(D)
```

Description

`TF = isLocked(D)` returns the locked status, `TF`, of the `DemosiacInterpolator` System object, `D`.

Introduced in R2015a

release

System object: visionhdl.DemosaicInterpolator
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

`release(D)`

Description

`release(D)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

D

`visionhdl.DemosaicInterpolator` System object

Introduced in R2015a

step

System object: visionhdl.DemosaicInterpolator
Package: visionhdl

Demosaic a Bayer pattern video stream

Syntax

`[pixelOut,ctrlOut] = step(D,pixelIn,ctrlIn)`

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[pixelOut,ctrlOut] = step(D,pixelIn,ctrlIn)` interpolates the missing color values of a Bayer pattern input pixel stream, and returns the next pixel value, `pixelOut`, as a vector of R'G'B' values. `pixelIn` represents one pixel in a Bayer pattern image.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

D

visionhdl.DemosaicInterpolator System object.

pixelIn

Single pixel, specified as a scalar value.

Supported data types:

- uint or int
- fixdt(0,N,0)
- double and single data types are supported for simulation but not for HDL code generation.

ctrlIn

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five logical signals. See "Pixel Control Structure".

Output Arguments

pixelOut

Single pixel, returned as a vector of three values in R'G'B' color space.

The data type of pixelOut is the same as the data type of pixelIn.

ctrlOut

Control signals indicating the validity of the output pixel and the location of the pixel within the frame, returned as a structure containing five logical signals. See "Pixel Control Structure".

Introduced in R2015a

visionhdl.Dilation System object

Package: visionhdl

Morphological dilation of binary pixel data

Description

`visionhdl.Dilation` replaces each pixel with the local maximum of the neighborhood around the pixel. The object operates on a stream of binary intensity values.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`D = visionhdl.Dilation` returns a System object, `D`, that performs morphological dilation on a binary video stream.

`D = visionhdl.Dilation(Name,Value)` returns a System object, `D`, with additional options specified by one or more `Name,Value` pair arguments. `Name` is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Properties not specified retain their default values.

Properties

`Neighborhood`

Pixel neighborhood, specified as a matrix or vector of ones and zeros.

The object supports neighborhoods of up to 32×32 pixels. To use a structuring element, specify `Neighborhood` as `getnhood(strel(`*`shape`*`))`.

Default: `[0,1,0;1,1,1;0,1,0]`

`LineBufferSize`

Size of the line memory buffer, specified as a scalar integer.

Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the object uses the next largest power of two. The object allocates (*neighborhood lines* – 1)-by-`LineBufferSize` memory locations to store the pixels.

Default: `2048`

Methods

clone	Create object having the same property values
isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics
step	Report dilated pixel value based on neighborhood

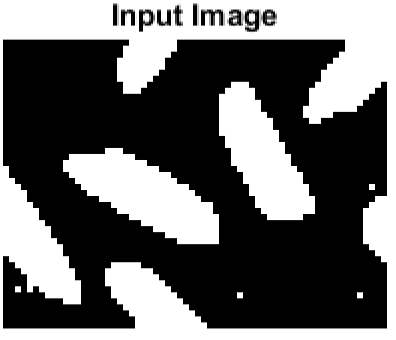
Examples

Morphological Dilate

Perform morphological dilate on a thumbnail image.

Load a source image from a file. Select a portion of the image that matches the desired test size. This source image contains `uint8` pixel intensity values. Apply a threshold to convert to binary pixel data.

```
frmOrig = imread('rice.png');
frmActivePixels = 64;
frmActiveLines = 48;
frmInput = frmOrig(1:frmActiveLines,1:frmActivePixels);
frmInput = frmInput>128;
figure
imshow(frmInput,'InitialMagnification',300)
title 'Input Image'
```



Create a serializer object and define inactive pixel regions. Make the number of inactive pixels following each active line at least double the horizontal size of the neighborhood. Make the number of lines following each frame at least double the vertical size of the neighborhood.

```
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines,...
      'TotalPixelsPerLine',frmActivePixels+20,...
      'TotalVideoLines',frmActiveLines+10,...
      'StartingActiveLine',3,...
```

```
      'FrontPorch',10);
```

Create a filter object.

```
 mdilate = visionhdl.Dilation( ...
      'Neighborhood',getnhood(strel('disk',3)));
```

Serialize the test image by calling `step` on the serializer object. `pixIn` is a vector of intensity values. `ctrlIn` is a vector of control signal structures.

```
[pixIn,ctrlIn] = step(frm2pix,frmInput);
```

Prepare to process pixels by preallocating output vectors.

```
[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
pixOut = false(numPixelsPerFrame,1);
ctrlOut  = repmat(pixelcontrolstruct,numPixelsPerFrame,1);
```

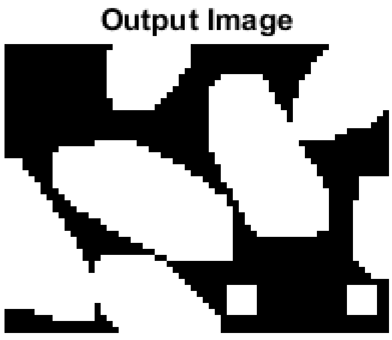
For each pixel in the padded frame, compute the morphed value. Monitor the control signals to determine latency of the object. The latency of a configuration depends on the number of active pixels in a line and the size of the neighborhood

```
foundValIn = false;
foundValOut = false;
for p = 1:numPixelsPerFrame
    if (ctrlIn(p).valid && foundValIn==0)
        foundValIn = p;
    end
    [pixOut(p),ctrlOut(p)] = step(mdilate,pixIn(p),ctrlIn(p));
    if (ctrlOut(p).valid && foundValOut==0)
        foundValOut = p;
    end
end
sprintf('object latency is %d cycles',foundValOut-foundValIn)
```

```
ans =

object latency is 186 cycles
```

Create a deserializer object with a format matching that of the serializer. Convert the pixel stream to an image frame by calling `step` on the deserializer object. Display the resulting image.

```
pix2frm = visionhdl.PixelsToFrame(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines);
[frmOutput,frmValid] = step(pix2frm,pixOut,ctrlOut);
if frmValid
    figure
    imshow(frmOutput, 'InitialMagnification',300)
    title 'Output Image'
end
```



Algorithm

This object implements the algorithms described on the `Dilation` block reference page.

See Also

`Dilation` | visionhdl.FrameToPixels | visionhdl.Erosion | vision.MorphologicalDilate | `imdilate`

Introduced in R2015a

clone

System object: visionhdl.Dilation
Package: visionhdl

Create object having the same property values

Syntax

`newD = clone(D)`

Description

`newD = clone(D)` creates another instance of the `Dilation` System object, `D`, that has the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

`D`

`visionhdl.Dilation` System object

Output Arguments

`newD`

New `Dilation` System object that has the same property values as the original System object.

Introduced in R2015a

isLocked

System object: visionhdl.Dilation
Package: visionhdl

Locked status (logical)

Syntax

```
TF = isLocked(D)
```

Description

`TF = isLocked(D)` returns the locked status, `TF`, of the `Dilation` System object, `D`.

Introduced in R2015a

release

System object: visionhdl.Dilation
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

`release(D)`

Description

`release(D)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

D

`visionhdl.Dilation` System object

Introduced in R2015a

step

System object: visionhdl.Dilation
Package: visionhdl

Report dilated pixel value based on neighborhood

Syntax

`[pixelOut,ctrlOut] = step(D,pixelIn,ctrlIn)`

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[pixelOut,ctrlOut] = step(D,pixelIn,ctrlIn)` returns the next pixel value, `pixelOut`, resulting from a morphological dilation operation on the neighborhood around each input pixel, `pixelIn`.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

D

`visionhdl.Dilation` System object

pixelIn

Single pixel, specified as a scalar `logical` value.

ctrlIn

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five `logical` signals. See "Pixel Control Structure".

Output Arguments

pixelOut

Single pixel transformed by a morphological operation, returned as a scalar `logical` value.

ctrlOut

Control signals indicating the validity of the output pixel and the location of the pixel within the frame, returned as a structure containing five `logical` signals. See "Pixel Control Structure".

Introduced in R2015a

visionhdl.GrayscaleDilation System object

Package: visionhdl

Morphological dilation of grayscale pixel data

Description

`visionhdl.GrayscaleDilation` performs morphological dilation on a stream of pixel intensity values. You can specify a neighborhood, or structuring element, of up to 32×32 pixels. For line, square, or rectangle structuring elements more than 8 pixels wide, the object uses the Van Herk algorithm to find the maximum. This algorithm uses only three comparators to find the maximums of all the rows, then uses a comparison tree to find the maximum of the row results.

For structuring elements less than 8 pixels wide, or that contain zero elements, the object implements a pipelined comparison tree for each row of the neighborhood. An additional comparison tree finds the maximum value of the row results. If the structuring element contains zeros that mask off pixels, the algorithm saves hardware resources by not implementing comparators for those pixel locations.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`D = visionhdl.GrayscaleDilation` returns a System object, `D`, that performs morphological dilation on a pixel stream.

`D = visionhdl.GrayscaleDilation(Name,Value)` returns a System object, `D`, with additional options specified by one or more `Name`,`Value` pair arguments. `Name` is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Properties not specified retain their default values.

Properties

`Neighborhood`

Pixel neighborhood, specified as a matrix or vector of ones and zeros.

The object supports neighborhoods of up to 32×32 pixels. To use a structuring element, specify `Neighborhood` as `getnhood(strel(`*`shape`*`))`. The minimum neighborhood size is a 2×2 matrix, or a 2×1 column vector. If the neighborhood is a row vector, it must be at least 8 columns wide and contain no zeros.

Default: `ones(5,5)`

`LineBufferSize`

Size of the line memory buffer, specified as a scalar integer.

Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the object uses the next largest power of two. The object allocates (*neighborhood lines* – 1)-by-`LineBufferSize` memory locations to store the pixels.

Default: `2048`

Methods

clone	Create object having the same property values
isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics

step	Report dilated pixel value based on neighborhood

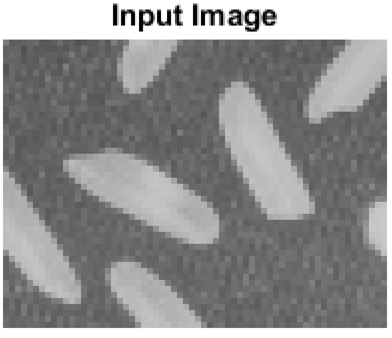
Examples

Grayscale Morphological Dilation

Perform morphological dilation on a grayscale thumbnail image.

Load a source image from a file. Select a portion of the image matching the desired test size.

```
frmOrig = imread('rice.png');
frmActivePixels = 64;
frmActiveLines = 48;
frmInput = frmOrig(1:frmActiveLines,1:frmActivePixels);
imshow(frmInput,'InitialMagnification',300)
title 'Input Image'
```



Create a serializer object and define the inactive pixel regions. Make the number of inactive pixels following each active line at least double the horizontal size of the neighborhood. Make the number of lines following each frame at least double the vertical size of the neighborhood.

```
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines,...
      'TotalPixelsPerLine',frmActivePixels+20,...
      'TotalVideoLines',frmActiveLines+20,...
      'StartingActiveLine',3,...
      'FrontPorch',10);
```

Create a filter object.

```
mdilate = visionhdl.GrayscaleDilation( ...
          'Neighborhood',ones(4,4));
```

Serialize the test image by calling `step` on the serializer object. `pixIn` is a vector of intensity values. `ctrlIn` is a vector of control signal structures.

```
[pixIn,ctrlIn] = step(frm2pix,frmInput);
```

Prepare to process pixels by preallocating output vectors.

```
[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
pixOut = uint8(zeros(numPixelsPerFrame,1));
ctrlOut = repmat(pixelcontrolstruct,numPixelsPerFrame,1);
```

For each pixel in the padded frame, compute the morphed value. Monitor the control signals to determine the latency of the object. The latency of a configuration depends on the number of active pixels in a line and the size of the neighborhood.

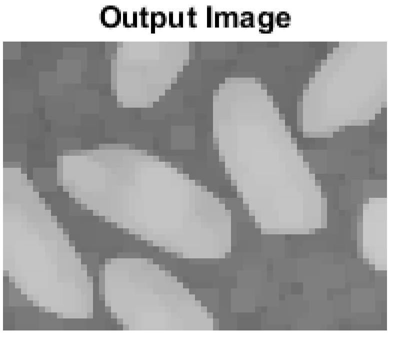
```
foundValIn = false;
foundValOut = false;
for p = 1:numPixelsPerFrame
    if (ctrlIn(p).valid && foundValIn==0)
        foundValIn = p;
    end
    [pixOut(p),ctrlOut(p)] = step(mdilate,pixIn(p),ctrlIn(p));
    if (ctrlOut(p).valid && foundValOut==0)
        foundValOut = p;
    end
end
sprintf('object latency is %d cycles',foundValOut-foundValIn)
```

```
ans =
```

```
object latency is 190 cycles
```

Create a deserializer object with a format matching that of the serializer. Convert the pixel stream to an image frame by calling `step` on the deserializer object. Display the resulting image.

```
pix2frm = visionhdl.PixelsToFrame(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines);
[frmOutput,frmValid] = step(pix2frm,pixOut,ctrlOut);
if frmValid
    figure
    imshow(frmOutput, 'InitialMagnification',300)
    title 'Output Image'
end
```



Algorithm

This object implements the algorithms described on the `Grayscale Dilation` block reference page.

See Also

vision.MorphologicalDilate | visionhdl.FrameToPixels | visionhdl.GrayscaleErosion | `Grayscale Dilation` | `imdilate`

Introduced in R2016a

clone

System object: visionhdl.GrayscaleDilation
Package: visionhdl

Create object having the same property values

Syntax

```
newD = clone(D)
```

Description

`newD = clone(D)` creates another instance of the `GrayscaleDilation` System object, `C`, that has the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

D

`visionhdl.GrayscaleDilation` System object

Output Arguments

newD

New `GrayscaleDilation` System object that has the same property values as the original System object.

Introduced in R2016a

isLocked

System object: visionhdl.GrayscaleDilation
Package: visionhdl

Locked status (logical)

Syntax

`TF = isLocked(D)`

Description

`TF = isLocked(D)` returns the locked status, `TF`, of the `GrayscaleDilation` System object, `D`.

Introduced in R2016a

release

System object: visionhdl.GrayscaleDilation
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

`release(D)`

Description

`release(D)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

D

`visionhdl.GrayscaleDilation` System object

Introduced in R2016a

step

System object: visionhdl.GrayscaleDilation
Package: visionhdl

Report dilated pixel value based on neighborhood

Syntax

`[pixelOut,ctrlOut] = step(D,pixelIn,ctrlIn)`

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[pixelOut,ctrlOut] = step(D,pixelIn,ctrlIn)` returns the next pixel value, `pixelOut`, resulting from morphological dilation on the neighborhood around each input pixel intensity value, `pixelIn`.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

D

visionhdl.GrayscaleDilation System object

pixelIn

Single pixel, specified as a scalar value.

Supported data types:

- uint8, uint16,uint32
- fixdt(0,N,M)
- double and single data types are supported for simulation but not for HDL code generation.

ctrlIn

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five logical signals. See "Pixel Control Structure".

Output Arguments

pixelOut

Single pixel transformed by a morphological operation, returned as a scalar value.

The data type is the same as the data type of pixelIn.

ctrlOut

Control signals indicating the validity of the output pixel and the location of the pixel within the frame, returned as a structure containing five logical signals. See "Pixel Control Structure".

Introduced in R2016a

visionhdl.EdgeDetector System object

Package: visionhdl

Find edges of objects

Description

`visionhdl.EdgeDetector` finds the edges in a grayscale pixel stream using the Sobel, Prewitt, or Roberts method. The object convolves the input pixels with derivative approximation matrices to find the gradient of pixel magnitude along two orthogonal directions. It then compares the sum of the squares of the gradients to a configurable threshold to determine if the gradients represent an edge. The Sobel and Prewitt methods calculate the gradient in horizontal and vertical directions. The Roberts method calculates the gradients at 45 and 135 degrees.

The object returns a binary image, as a stream of pixel values. A pixel value of 1 indicates that the pixel is an edge. You can optionally enable output of the gradient values in the two orthogonal directions at each pixel.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`ED = visionhdl.EdgeDetector` returns a System object, `ED`, that detects edges using the Sobel method.

`ED = visionhdl.EdgeDetector(Name,Value)` returns a System object, `ED`, with additional options specified by one or more `Name`,`Value` pair arguments. `Name` is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Properties not specified retain their default values.

Properties

`Method`

Edge detection algorithm.

Specify 'Sobel', 'Prewitt', or 'Roberts' method.

Default: 'Sobel'

`BinaryImageOutputPort`

Enable the Edge output of the step method.

When this property is true, the step method returns a binary pixel value representing whether the object detected an edge at each location in the frame.

Default: true

`GradientComponentOutputPorts`

Enable the G1 and G2 outputs of the step method.

When this property is true, the step method returns two values representing the gradients calculated in two orthogonal directions at each pixel. Set the data type for this argument in the GradientDataType property.

Default: false

`ThresholdSource`

Source for the gradient threshold value that indicates an edge.

Set this property to 'Input port' to set the threshold as an input argument to the step method. When this property is set to 'Property', set the threshold in the Threshold property.

Default: 'Property'

`Threshold`

Gradient threshold value that indicates an edge, specified as a numeric scalar value.

The object compares the square of this to the sum of the squares of the gradients. The object casts this value to the data type of the gradients. This property applies when you set `ThresholdSource` to '`Property`'.

Default: 20

`LineBufferSize`

Size of the line memory buffer, specified as a scalar integer.

Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the object uses the next largest power of two. The object allocates (N - 1)-by-`LineBufferSize` memory locations to store the pixels, where N is the number of lines in the differential approximation matrix. If you set the `Method` property to '`Sobel`' or '`Prewitt`', then N is 3. If you set the `Method` property to '`Roberts`', then N is 2.

Default: 2048

`RoundingMethod`

Rounding mode used for fixed-point operations.

The object uses fixed-point arithmetic for internal calculations when the input is any integer or fixed-point data type. This option does not apply when the input data type is `single` or `double`.

Default: `Floor`

`OverflowAction`

Overflow action used for fixed-point operations.

The object uses fixed-point arithmetic for internal calculations when the input is any integer or fixed-point data type. This option does not apply when the input data type is `single` or `double`.

Default: `Wrap`

`GradientDataType`

Data type for the gradient output values, specified as `numerictype(signed,WL,FL)`, where WL is the word length and FL is the fraction length in bits.

- 'Full precision' (default) — Use full-precision rules based on the data type of the pixelIn argument of the step method, and the coefficients of the derivative approximation matrices.
- 'custom' — Use the data type defined in theCustomGradientDataType property.

CustomGradientDataType

Data type for the gradient output values, specified as numerictype(signed,WL,FL), where *WL* is the word length and *FL* is the fraction length in bits.

Default: numerictype(1,8,0)

Methods

clone	Create object having the same property values
isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics
step	Detect edges at an image pixel

Examples

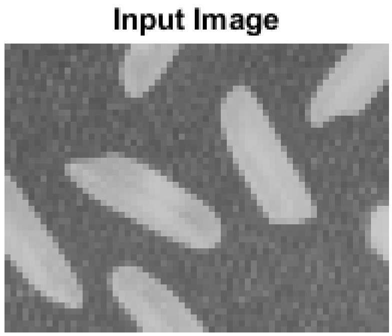
Edge Detection Using Sobel Method

Detect edges in a thumbnail image using the Sobel method.

Prepare a test image by selecting a portion of an image file.

```
frmActivePixels = 64;
frmActiveLines = 48;
frmOrig = imread('rice.png');
frmInput = frmOrig(1:frmActiveLines,1:frmActivePixels);
figure
imshow(frmInput,'InitialMagnification',300)
```

```
title 'Input Image'
```



Create a serializer and specify the size of inactive pixel regions.

```
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines,...
      'TotalPixelsPerLine',frmActivePixels+10,...
      'TotalVideoLines',frmActiveLines+10,...
      'StartingActiveLine',6,...
      'FrontPorch',5);
```

Create an edge detection object with the default property values. The default detection method is Sobel.

```
edgeDetectSobel = visionhdl.EdgeDetector();
```

Serialize the test image using the object you created. `pixIn` is a vector of intensity values. `ctrlIn` is a vector of control signal structures. Preallocate vectors for the output signals.

```
[pixIn,ctrlIn] = step(frm2pix,frmInput);

[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
```

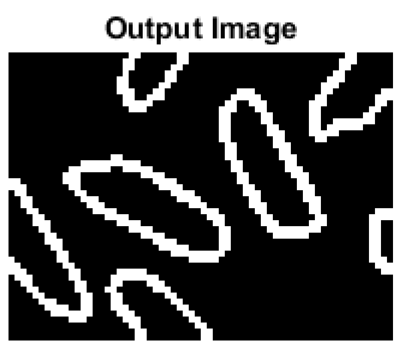
```
ctrlOut = repmat(pixelcontrolstruct,numPixelsPerFrame,1);
edgeOut = false(numPixelsPerFrame,1);
```

For each pixel in the stream, compute whether it represents an edge.

```
for p = 1:numPixelsPerFrame
   [edgeOut(p),ctrlOut(p)] = step(edgeDetectSobel,pixIn(p),ctrlIn(p));
end
```

Create a deserializer with a format matching that of the serializer. Use the deserializer to convert the output pixel stream to an image frame.

```
pix2frm = visionhdl.PixelsToFrame(...
       'NumComponents',1,...
       'VideoFormat','custom',...
       'ActivePixelsPerLine',frmActivePixels,...
       'ActiveVideoLines',frmActiveLines);

[frmOutput,frmValid] = step(pix2frm,edgeOut,ctrlOut);
if frmValid
    figure
    imshow(frmOutput, 'InitialMagnification',300)
    title 'Output Image'
end
```



- “Pixel-Streaming Design in MATLAB”

Algorithm

This object implements the algorithms described on the `Edge Detector` block reference page.

See Also

vision.EdgeDetector | visionhdl.FrameToPixels | `edge` | `Edge Detector`

Introduced in R2015a

clone

System object: visionhdl.EdgeDetector
Package: visionhdl

Create object having the same property values

Syntax

`newED = clone(ED)`

Description

`newED = clone(ED)` creates another instance of the `EdgeDetector` System object, `ED`, that has the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

`ED`

`visionhdl.EdgeDetector` System object

Output Arguments

`newED`

New `EdgeDetector` System object that has the same property values as the original System object.

Introduced in R2015a

isLocked

System object: visionhdl.EdgeDetector
Package: visionhdl

Locked status (logical)

Syntax

`TF = isLocked(ED)`

Description

`TF = isLocked(ED)` returns the locked status, `TF`, of the `EdgeDetector` System object, `ED`.

Introduced in R2015a

release

System object: visionhdl.EdgeDetector
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

`release(ED)`

Description

`release(ED)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

ED

`visionhdl.EdgeDetector` System object

Introduced in R2015a

step

System object: visionhdl.EdgeDetector
Package: visionhdl

Detect edges at an image pixel

Syntax

```
[edge,ctrlOut] = step(ED,pixelIn,ctrlIn)
[G1,G2,ctrlOut] = step(ED,pixelIn,ctrlIn)
[edge,ctrlOut] = step(ED,pixelIn,ctrlIn,thresh)
```

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[edge,ctrlOut] = step(ED,pixelIn,ctrlIn)` detects edges in the neighborhood of `pixelIn` by computing the gradient in two orthogonal directions. The `edge` output argument is a binary value indicating whether the sum of the squares of the gradients for the input pixel is above the threshold indicating an edge.

`[G1,G2,ctrlOut] = step(ED,pixelIn,ctrlIn)` detects edges in the neighborhood of `pixelIn` by computing the gradient in two orthogonal directions. Use this syntax when you set `GradientComponentOutputPorts` property to `true`. The `G1` and `G2` output arguments are the gradients calculated in the two orthogonal directions. When you set the `Method` property to `'Sobel'` or `'Prewitt'`, the first argument is the vertical gradient, and the second argument is the horizontal gradient. When you set the `Method` property to `'Roberts'`, the first argument is the 45 degree gradient, and the second argument is the 135 degree gradient.

`[edge,ctrlOut] = step(ED,pixelIn,ctrlIn,thresh)` detects edges in the neighborhood of `pixelIn` by computing the gradient in two orthogonal directions. Use this syntax when you set `ThresholdSource` property to `'InputPort'`. The `edge` output argument is a binary value indicating whether the sum of the squares of the gradients was above the threshold, `thresh`, squared.

You can use any combination of the optional port syntaxes.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

`ED`

`visionhdl.EdgeDetector` System object.

`pixelIn`

Intensity of a single pixel, specified as a scalar value.

Supported data types:

- `uint` or `int`
- `fixdt()`
- `double` and `single` data types are supported for simulation but not for HDL code generation.

ctrlIn

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five `logical` signals. See “Pixel Control Structure”.

thresh

Gradient threshold value that indicates an edge, specified as a scalar numeric value.

The object compares this value squared to the sum of the squares of the gradients. This argument is accepted when you set `ThresholdSource` property to `'InputPort'`.

Output Arguments

edge

Pixel value indicating an edge at this pixel, returned as a scalar binary value.

G1

Gradient calculated in the first direction, returned as a scalar value.

This argument is returned when you set `GradientComponentOutputPorts` property to `true`. If you set the `Method` property to `'Sobel'` or `'Prewitt'`, this argument is the vertical gradient. When you set the `Method` property to `'Roberts'`, this argument is the 45 degree gradient.

Configure the data type of the gradients by using the `GradientComponentDataType` and `CustomGradientComponent` properties.

G2

Gradient calculated in the second direction, returned as a scalar value.

This argument is returned when you set `GradientComponentOutputPorts` property to `true`. If you set the `Method` property to `'Sobel'` or `'Prewitt'`, this argument is the horizontal gradient. When you set the `Method` property to `'Roberts'`, this argument is the 135 degree gradient.

Configure the data type of the gradients by using the `GradientComponentDataType` and `CustomGradientComponent` properties.

`ctrlOut`

Control signals indicating the validity of the output pixel and the location of the pixel within the frame, returned as a structure containing five `logical` signals. See “Pixel Control Structure”.

Introduced in R2015a

visionhdl.Erosion System object

Package: visionhdl

Morphological erosion of binary pixel data

Description

`visionhdl.Erosion` replaces each pixel with the local minimum of the neighborhood around the pixel. The object operates on a stream of binary intensity values.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`E = visionhdl.Erosion` returns a System object, `E`, that performs morphological erosion on a binary pixel stream.

`E = visionhdl.Erosion(Name,Value)` returns a System object, `E`, with additional options specified by one or more `Name`,`Value` pair arguments. `Name` is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Properties not specified retain their default values.

Properties

`Neighborhood`

Pixel neighborhood, specified as a matrix or vector of ones and zeros.

The object supports neighborhoods of up to 32×32 pixels. To use a structuring element, specify `Neighborhood` as `getnhood(strel(`*`shape`*`))`.

Default: `ones(3,3)`

`LineBufferSize`

Size of the line memory buffer, specified as a scalar integer.

Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the object uses the next largest power of two. The object allocates (*neighborhood lines* – 1)-by-`LineBufferSize` memory locations to store the pixels.

Default: `2048`

Methods

clone	Create object having the same property values
isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics
step	Report eroded pixel value based on neighborhood

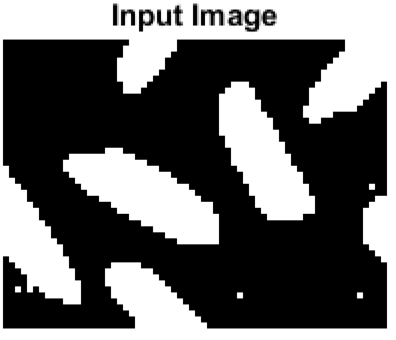
Examples

Morphological Erode

Perform morphological erode on a thumbnail image.

Load a source image from a file. Select a portion of the image that matches the desired test size. This source image contains `uint8` pixel intensity values. Apply a threshold to convert to binary pixel data.

```
frmOrig = imread('rice.png');
frmActivePixels = 64;
frmActiveLines = 48;
frmInput = frmOrig(1:frmActiveLines,1:frmActivePixels);
frmInput = frmInput>128;
figure
imshow(frmInput,'InitialMagnification',300)
title 'Input Image'
```



Create a serializer object and define inactive pixel regions. Make the number of inactive pixels following each active line at least double the horizontal size of the neighborhood. Make the number of lines following each frame at least double the vertical size of the neighborhood.

```
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines,...
      'TotalPixelsPerLine',frmActivePixels+20,...
      'TotalVideoLines',frmActiveLines+10,...
      'StartingActiveLine',3,...
```

```
'FrontPorch',10);
```

Create a filter object.

```
merode = visionhdl.Erosion( ...
         'Neighborhood',ones(2,7));
```

Serialize the test image by calling `step` on the serializer object. `pixIn` is a vector of intensity values. `ctrlIn` is a vector of control signal structures.

```
[pixIn,ctrlIn] = step(frm2pix,frmInput);
```

Prepare to process pixels by preallocating output vectors.

```
[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
pixOut = false(numPixelsPerFrame,1);
ctrlOut  = repmat(pixelcontrolstruct,numPixelsPerFrame,1);
```

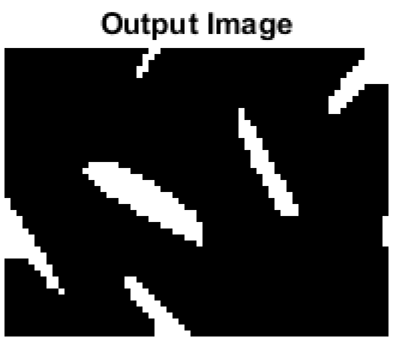
For each pixel in the padded frame, compute the morphed value. Monitor the control signals to determine latency of the object. The latency of a configuration depends on the number of active pixels in a line and the size of the neighborhood

```
foundValIn = false;
foundValOut = false;
for p = 1:numPixelsPerFrame
    if (ctrlIn(p).valid && foundValIn==0)
        foundValIn = p;
    end
    [pixOut(p),ctrlOut(p)] = step(merode,pixIn(p),ctrlIn(p));
    if (ctrlOut(p).valid && foundValOut==0)
        foundValOut = p;
    end
end
sprintf('object latency is %d cycles',foundValOut-foundValIn)
```

```
ans =

object latency is 105 cycles
```

Create a deserializer object with a format matching that of the serializer. Convert the pixel stream to an image frame by calling `step` on the deserializer object. Display the resulting image.

```
pix2frm = visionhdl.PixelsToFrame(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines);
[frmOutput,frmValid] = step(pix2frm,pixOut,ctrlOut);
if frmValid
    figure
    imshow(frmOutput, 'InitialMagnification',300)
    title 'Output Image'
end
```



Algorithm

This object implements the algorithms described on the `Erosion` block reference page.

See Also

`Erosion` | vision.MorphologicalDilate | visionhdl.FrameToPixels | vision.MorphologicalErode | `imerode`

Introduced in R2015a

clone

System object: visionhdl.Erosion
Package: visionhdl

Create object having the same property values

Syntax

`newE = clone(E)`

Description

`newE = clone(E)` creates another instance of the `Erosion` System object, `E`, that has the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

E

`visionhdl.Erosion` System object

Output Arguments

newE

New `Erosion` System object that has the same property values as the original System object. The new unlocked object contains uninitialized states.

Introduced in R2015a

isLocked

System object: visionhdl.Erosion
Package: visionhdl

Locked status (logical)

Syntax

```
TF = isLocked(E)
```

Description

`TF = isLocked(E)` returns the locked status, `TF`, of the `DemosiacInterpolator` System object, `E`.

Introduced in R2015a

release

System object: visionhdl.Erosion
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

`release(E)`

Description

`release(E)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

E

`visionhdl.Erosion` System object

Introduced in R2015a

step

System object: visionhdl.Erosion
Package: visionhdl

Report eroded pixel value based on neighborhood

Syntax

`[pixelOut,ctrlOut] = step(E,pixelIn,ctrlIn)`

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[pixelOut,ctrlOut] = step(E,pixelIn,ctrlIn)` returns the next pixel value, `pixelOut`, in the pixel stream resulting from a morphological erosion operation on the neighborhood around each input pixel, `pixelIn`.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

E

visionhdl.Erosion System object

pixelIn

Single pixel, specified as a scalar logical value.

ctrlIn

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five logical signals. See "Pixel Control Structure".

Output Arguments

pixelOut

Single pixel transformed by a morphological operation, returned as a scalar logical value.

ctrlOut

Control signals indicating the validity of the output pixel and the location of the pixel within the frame, returned as a structure containing five logical signals. See "Pixel Control Structure".

Introduced in R2015a

visionhdl.GrayscaleErosion System object

Package: visionhdl

Morphological erosion of grayscale pixel data

Description

`visionhdl.GrayscaleErosion` performs morphological erosion on a stream of pixel intensity values. You can specify a neighborhood, or structuring element, of up to 32×32 pixels. For line, square, or rectangle structuring elements more than 8 pixels wide, the object uses the Van Herk algorithm to find the maximum. This algorithm uses only three comparators to find the maximums of all the rows, then uses a comparison tree to find the maximum of the row results.

For structuring elements less than 8 pixels wide, or that contain zero elements, the object implements a pipelined comparison tree for each row of the neighborhood. An additional comparison tree finds the maximum value of the row results. If the structuring element contains zeros that mask off pixels, the algorithm saves hardware resources by not implementing comparators for those pixel locations.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`E = visionhdl.GrayscaleErosion` returns a System object, `E`, that performs a morphological erosion on a pixel stream.

`E = visionhdl.GrayscaleErosion(Name,Value)` returns a System object, `E`, with additional options specified by one or more `Name`,`Value` pair arguments. `Name` is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Properties not specified retain their default values.

Properties

`Neighborhood`

Pixel neighborhood, specified as a matrix or vector of ones and zeros.

The object supports neighborhoods of up to 32×32 pixels. To use a structuring element, specify `Neighborhood` as `getnhood(strel(`*`shape`*`))`. The minimum neighborhood size is a 2×2 matrix, or a 2×1 column vector. If the neighborhood is a row vector, it must be at least 8 columns wide and contain no zeros.

Default: `ones(3,3)`

`LineBufferSize`

Size of the line memory buffer, specified as a scalar integer.

Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the object uses the next largest power of two. The object allocates (*neighborhood lines* – 1)-by-`LineBufferSize` memory locations to store the pixels.

Default: `2048`

Methods

clone	Create object having the same property values
isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics

step	Report eroded pixel value based on neighborhood

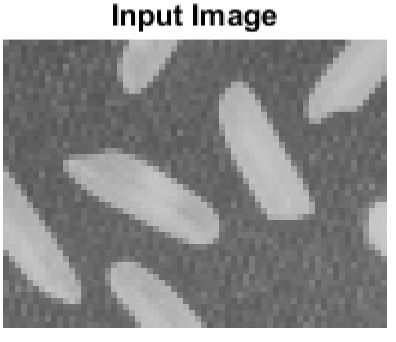
Examples

Grayscale Morphological Erosion

Perform morphological erosion on a grayscale thumbnail image.

Load a source image from a file. Select a portion of the image matching the desired test size.

```
frmOrig = imread('rice.png');
frmActivePixels = 64;
frmActiveLines = 48;
frmInput = frmOrig(1:frmActiveLines,1:frmActivePixels);
imshow(frmInput,'InitialMagnification',300)
title 'Input Image'
```



Create a serializer object and define the inactive pixel regions. Make the number of inactive pixels following each active line at least double the horizontal size of the neighborhood. Make the number of lines following each frame at least double the vertical size of the neighborhood.

```
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines,...
      'TotalPixelsPerLine',frmActivePixels+20,...
      'TotalVideoLines',frmActiveLines+20,...
      'StartingActiveLine',3,...
      'FrontPorch',10);
```

Create a filter object.

```
merode = visionhdl.GrayscaleErosion( ...
          'Neighborhood',ones(2,5));
```

Serialize the test image by calling `step` on the serializer object. `pixIn` is a vector of intensity values. `ctrlIn` is a vector of control signal structures.

```
[pixIn,ctrlIn] = step(frm2pix,frmInput);
```

Prepare to process pixels by preallocating output vectors.

```
[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
pixOut = uint8(zeros(numPixelsPerFrame,1));
ctrlOut = repmat(pixelcontrolstruct,numPixelsPerFrame,1);
```

For each pixel in the padded frame, compute the morphed value. Monitor the control signals to determine the latency of the object. The latency of a configuration depends on the number of active pixels in a line and the size of the neighborhood.

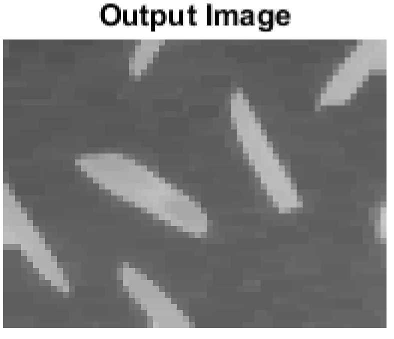
```
foundValIn = false;
foundValOut = false;
for p = 1:numPixelsPerFrame
    if (ctrlIn(p).valid && foundValIn==0)
        foundValIn = p;
    end
    [pixOut(p),ctrlOut(p)] = step(merode,pixIn(p),ctrlIn(p));
    if (ctrlOut(p).valid && foundValOut==0)
        foundValOut = p;
    end
end
sprintf('object latency is %d cycles',foundValOut-foundValIn)
```

```
ans =
```

```
object latency is 109 cycles
```

Create a deserializer object with a format matching that of the serializer. Convert the pixel stream to an image frame by calling `step` on the deserializer object. Display the resulting image.

```
pix2frm = visionhdl.PixelsToFrame(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines);
[frmOutput,frmValid] = step(pix2frm,pixOut,ctrlOut);
if frmValid
    figure
    imshow(frmOutput, 'InitialMagnification',300)
    title 'Output Image'
end
```



Algorithm

This object implements the algorithms described on the `Grayscale Erosion` block reference page.

See Also

visionhdl.FrameToPixels | vision.MorphologicalErode | `imerode` |
visionhdl.GrayscaleDilation | `Grayscale Erosion`

Introduced in R2016a

clone

System object: visionhdl.GrayscaleErosion
Package: visionhdl

Create object having the same property values

Syntax

`newE = clone(E)`

Description

`newE = clone(E)` creates another instance of the `GrayscaleErosion` System object, `E`, that has the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

E

`visionhdl.GrayscaleErosion` System object

Output Arguments

newE

New `GrayscaleErosion` System object that has the same property values as the original System object.

Introduced in R2016a

isLocked

System object: visionhdl.GrayscaleErosion
Package: visionhdl

Locked status (logical)

Syntax

`TF = isLocked(E)`

Description

`TF = isLocked(E)` returns the locked status, `TF`, of the `GrayscaleErosion` System object, `E`.

Introduced in R2016a

release

System object: visionhdl.GrayscaleErosion
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

`release(E)`

Description

`release(E)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

E

`visionhdl.GrayscaleErosion` System object

Introduced in R2016a

step

System object: visionhdl.GrayscaleErosion
Package: visionhdl

Report eroded pixel value based on neighborhood

Syntax

```
[pixelOut,ctrlOut] = step(E,pixelIn,ctrlIn)
```

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[pixelOut,ctrlOut] = step(E,pixelIn,ctrlIn)` returns the next pixel value, `pixelOut`, resulting from a morphological erosion of the neighborhood around each input pixel intensity value, `pixelIn`.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

E

`visionhdl.GrayscaleErosion` System object

pixelIn

Single pixel, specified as a scalar value.

Supported data types:

- `uint8`, `uint16`,`uint32`
- `fixdt(0,N,M)`
- `double` and `single` data types are supported for simulation but not for HDL code generation.

ctrlIn

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five `logical` signals. See “Pixel Control Structure”.

Output Arguments

pixelOut

Single pixel transformed by a morphological operation, returned as a scalar value.

The data type is the same as the data type of `pixelIn`.

ctrlOut

Control signals indicating the validity of the output pixel and the location of the pixel within the frame, returned as a structure containing five `logical` signals. See “Pixel Control Structure”.

Introduced in R2016a

visionhdl.FrameToPixels System object

Package: visionhdl

Convert frame-based video to pixel stream

Description

`visionhdl.FrameToPixels` converts color or grayscale frame-based video to a pixel stream and control structure. The control structure indicates the validity of each pixel and its location in the frame. The pixel stream format can include padding pixels around the active frame. You can configure the frame and padding dimensions by selecting a common video format or specifying custom dimensions. See "Streaming Pixel Interface" for details of the pixel stream format.

Use this object to generate input for a function targeted for HDL code generation. This block does not support HDL code generation.

If your design converts frames to a pixel stream and later converts the stream back to frames, specify the same video format for the `FrameToPixels` object and the `PixelsToFrame` object.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`F2P = visionhdl.FrameToPixels` returns a System object, `F2P`, that serializes a grayscale 1080×1920 frame into a 1080p pixel stream with standard padding around the active data.

`F2P = visionhdl.FrameToPixels(Name,Value)` returns a System object, `F2P`, with additional options specified by one or more `Name,Value` pair arguments. `Name`

is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Properties not specified retain their default values.

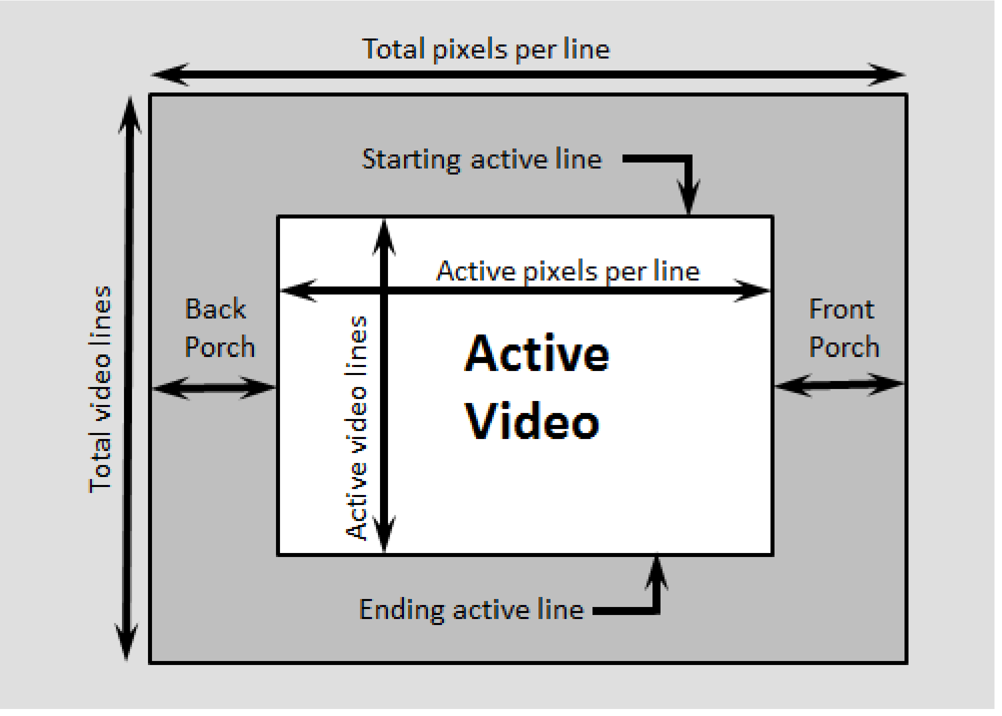
Properties

`NumComponents`

Components of each pixel, specified as 1, 3, or 4. Set to 1 for grayscale video. Set to 3 for color video, for example, {R,G,B} or {Y,Cb,Cr}. Set to 4 to use color with an alpha channel for transparency. The `step` method returns a *P*-by-`NumComponents` matrix, where *P* is the total number of pixels. The default is 1.

`VideoFormat`

Dimensions of active and inactive regions of a video frame. To select a predefined format, specify the `VideoFormat` property as one of the options in the first column of the table. For a custom format, set `VideoFormat` to `'Custom'`, and specify the dimension properties as integers. The frame dimensions are indicated in the diagram.



Video Format	Active Pixels Per Line	Active Video Lines	Total Pixels Per Line	Total Video Lines	Starting Active Line	Front Porch
240p	320	240	402	324	1	44
480p	640	480	800	525	36	16
480pH	720	480	858	525	33	16
576p	720	576	864	625	47	12
720p	1280	720	1650	750	25	110
768p	1024	768	1344	806	10	24
1024p	1280	1024	1688	1066	42	48
1080p (default)	1920	1080	2200	1125	42	88
1200p	1600	1200	2160	1250	50	64

Video Format	Active Pixels Per Line	Active Video Lines	Total Pixels Per Line	Total Video Lines	Starting Active Line	Front Porch
2KCinema	2048	1080	2750	1125	42	639
4KUHDTV	3840	2160	4400	2250	42	88
8KUHDTV	7680	4320	8800	4500	42	88
Custom	User-defined	User-defined	User-defined	User-defined	User-defined	User-defined

Note: When using a custom format, the properties you enter for the active and inactive dimensions of the image must add up to the total frame dimensions.

For the horizontal direction, `TotalPixelsPerLine` must be greater than or equal to `FrontPorch` + `ActivePixelsPerLine`. The block calculates *BackPorch* = `TotalPixelsPerLine` − `FrontPorch` − `ActivePixelsPerLine`.

For the vertical direction, `TotalVideoLines` must be greater than or equal to `StartingActiveLine` + `ActiveVideoLines` − 1. The block calculates *EndingActiveLine* = `StartingActiveLine` + `ActiveVideoLines` − 1.

If you specify a format that does not conform to these rules, the object reports an error.

Note: When using a custom format, `ActivePixelsPerLine` must be greater than 1. Also, set the horizontal blanking interval, or `BackPorch` + `FrontPorch`, according to these guidelines.

- The total of `BackPorch` + `FrontPorch` must be at least 2 times the largest *kernel size* of the algorithm in the objects following the `visionhdl.FrameToPixels` object. If the *kernel size* is < 4, the total porch must be at least 8 pixels.
- The `BackPorch` must be at least 6 pixels. This parameter is the number of inactive pixels before the first valid pixel in a frame.

Methods

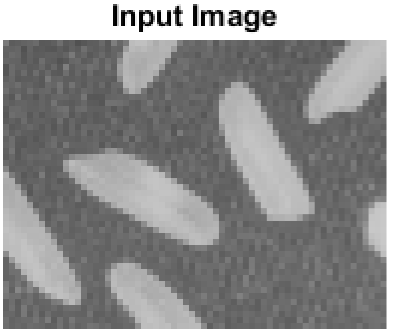
clone	Create object having the same property values
isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics
step	Convert image frame to pixel stream

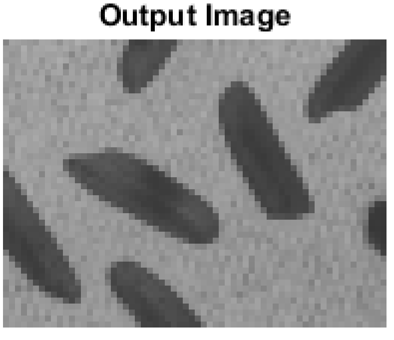
Examples

Convert Between Full-Frame and Pixel-Streaming Data

```
% This example converts a custom-size grayscale image to a pixel stream. It
% uses the visionhdl.LookupTable object to obtain the negative image. Then
% it converts the pixel-stream back to a full-frame image.

% Set the dimensions of the test image
frmActivePixels = 64;
frmActiveLines = 48;

% Load the source image
frmOrig = imread('rice.png');

% Select a portion of the image matching the desired test size
frmInput = frmOrig(1:frmActiveLines,1:frmActivePixels);
figure
imshow(frmInput,'InitialMagnification',300)
title 'Input Image'

% Create serializer and specify size of inactive pixel regions
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines,...
      'TotalPixelsPerLine',frmActivePixels+10,...
      'TotalVideoLines',frmActiveLines+10,...
      'StartingActiveLine',6,...
      'FrontPorch',5);
```

```
% Create LUT to generate the negative of the input image
tabledata = linspace(255,0,256);
inverter = visionhdl.LookupTable(tabledata);

% Serialize the test image
% pixIn is a vector of intensity values
% ctrlIn is a vector of control signal structures
[pixIn,ctrlIn] = step(frm2pix,frmInput);

% Prepare to process pixels
[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
pixOut = zeros(numPixelsPerFrame,1,'uint8');
ctrlOut = repmat(pixelcontrolstruct,numPixelsPerFrame,1);

% For each pixel in the stream, look up the negative of the pixel value
for p = 1:numPixelsPerFrame
    [pixOut(p),ctrlOut(p)] = step(inverter,pixIn(p),ctrlIn(p));
end

% Create deserializer with format matching that of the serializer
pix2frm = visionhdl.PixelsToFrame(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines);

% Convert the pixel stream to an image frame
[frmOutput,frmValid] = step(pix2frm,pixOut,ctrlOut);
if frmValid
    figure
    imshow(frmOutput,'InitialMagnification',300)
    title 'Output Image'
end
```





- “Pixel-Streaming Design in MATLAB”

See Also

visionhdl.PixelsToFrame | `Frame To Pixels`

More About

- “Streaming Pixel Interface”

Introduced in R2015a

clone

System object: visionhdl.FrameToPixels
Package: visionhdl

Create object having the same property values

Syntax

```
newF2P = clone(F2P)
```

Description

`newF2P = clone(F2P)` creates another instance of the `FrameToPixels` System object, `F2P`, that has the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

`F2P`

`visionhdl.FrameToPixels` System object

Output Arguments

`newF2P`

New `FrameToPixels` System object that has the same property values as the original object.

Introduced in R2015a

isLocked

System object: visionhdl.FrameToPixels
Package: visionhdl

Locked status (logical)

Syntax

`TF = isLocked(F2P)`

Description

`TF = isLocked(F2P)` returns the locked status, `TF`, of the `FrameToPixels` System object, `F2P`.

Introduced in R2015a

release

System object: visionhdl.FrameToPixels
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

`release(F2P)`

Description

`release(F2P)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

`F2P`

`visionhdl.FrameToPixels` System object

Introduced in R2015a

step

System object: visionhdl.FrameToPixels
Package: visionhdl

Convert image frame to pixel stream

Syntax

```
[pixels,ctrlOut] = step(F2P,frm)
```

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

```
[pixels,ctrlOut] = step(F2P,frm)
```

Converts the input image matrix, `frm`, to a vector of pixel values, `pixels`, and an associated vector of control structures, `ctrlOut`. The control structure indicates the validity of each pixel and its location in the frame. The output pixels include padding around the active image, specified by the `VideoFormat` property.

See “Streaming Pixel Interface” for details of the pixel stream format.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

F2P

visionhdl.FrameToPixels System object

frm

Input image, specified as an ActiveVideoLines-by-ActivePixelsPerLine-by-NumComponents matrix, where:

- ActiveVideoLines is the height of the active image
- ActivePixelsPerLine is the width of the active image
- NumComponents is the number of components used to express a single pixel

Set the size of the active image using the VideoFormat property. If the dimensions of im do not match that specified by VideoFormat, the object returns a warning.

Supported data types:

- uint or int
- fixdt()
- logical
- double or single

Output Arguments

pixels

Pixel values, returned as a *P*-by-NumComponents matrix, where:

- *P* is the total number of pixels in the padded image, calculated as TotalPixelsPerLine × TotalVideoLines
- NumComponents is the number of components used to express a single pixel

Set the size of the padded image using the VideoFormat property. The data type of the pixel values is the same as im.

`ctrlOut`

Control structures associated with the output pixels, returned as a *P*-by-1 vector. *P* is the total number of pixels in the padded image, calculated as `TotalPixelsPerLine` × `TotalVideoLines`. Each structure contains five control signals indicating the validity of the pixel and its location in the frame. See “Pixel Control Structure”.

Introduced in R2015a

visionhdl.GammaCorrector System object

Package: visionhdl

Apply or remove gamma correction

Description

`visionhdl.GammaCorrector` applies or removes gamma correction on a stream of pixels. Gamma correction adjusts linear pixel values so that the modified values fit a curve. The de-gamma operation performs the opposite operation to obtain linear pixel values.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`G = visionhdl.GammaCorrector` returns a System object that applies or removes gamma correction on a stream of pixels.

`G = visionhdl.GammaCorrector(Name,Value)` returns a System object, `G`, with additional options specified by one or more `Name,Value` pair arguments. `Name` is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Properties not specified retain their default values.

`G = visionhdl.GammaCorrector(operation,gammaValue,Name,Value)` returns a System object with the `Correction` property set to `operation`, the `Gamma` property set to `gammaValue`, and additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

operation

Type of correction, specified as either `Gamma` or `De-gamma`. This argument sets the `Correction` property value.

gammaValue

Target or current gamma value, specified as a scalar value greater than or equal to 1. This argument sets the Gamma property value.

Output Arguments

G

`visionhdl.GammaCorrector` System object

Properties

Correction

Direction of intensity curve adjustment

- `Gamma` (default) — Apply gamma correction.
- `De-gamma` — Remove gamma correction.

Gamma

Target or current gamma value, specified as a scalar greater than or equal to 1.

- When you set `Correction` to `Gamma`, set this property to the target gamma value of the output video stream.
- When you set `Correction` to `De-gamma`, set this property to the gamma value of the input video stream.

Default: 2.2

LinearSegment

Option to include a linear segment in the gamma curve, specified as a `logical` value. When you set this property to `true`, the gamma curve has a linear portion near the origin.

Default: `true`

BreakPoint

Pixel value that corresponds to the point where the gamma curve and linear segment meet. Specify `Breakpoint` as a scalar value between 0 and 1, exclusive. This property applies only when the `LinearSegment` property is set to `true`.

Default: 0.018

Methods

clone	Create object with same property values
isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics
step	Apply or remove gamma correction on one pixel

Examples

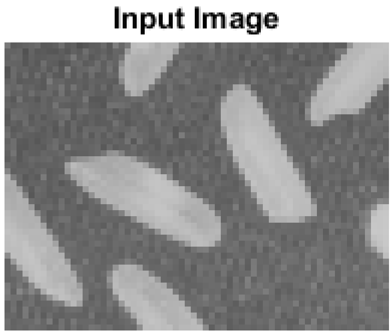
Gamma Correction

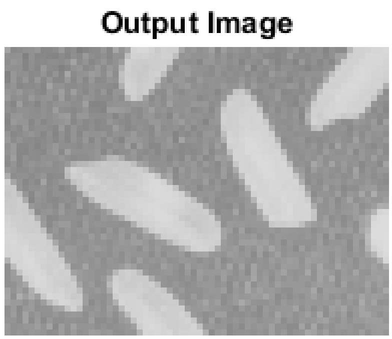
This example performs gamma correction on a thumbnail image.

```
% Set dimensions of the test image
frmActivePixels = 64;
frmActiveLines = 48;

% Load the source image
frmOrig = imread('rice.png');

% Select a portion of the image matching the desired test size
frmInput = frmOrig(1:frmActiveLines,1:frmActivePixels);
```

```
figure
imshow(frmInput,'InitialMagnification',300)
title 'Input Image'

% Create serializer and specify size of inactive pixel regions
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines,...
      'TotalPixelsPerLine',frmActivePixels+10,...
      'TotalVideoLines',frmActiveLines+10,...
      'StartingActiveLine',6,...
      'FrontPorch',5);

% Create gamma corrector object
 gammacorr = visionhdl.GammaCorrector(...
      'Gamma', 1.75);

% Serialize the test image
% pixIn is a vector of intensity values
% ctrlIn is a vector of control signal structures
[pixIn,ctrlIn] = step(frm2pix,frmInput);

% Prepare to process pixels
[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
pixOut = zeros(numPixelsPerFrame,1,'uint8');
ctrlOut  = repmat(pixelcontrolstruct,numPixelsPerFrame,1);

% For each pixel in the stream, compute the gamma corrected pixel value
for p = 1:numPixelsPerFrame
    [pixOut(p),ctrlOut(p)] = step(gammacorr,pixIn(p),ctrlIn(p));
end

% Create deserializer with format matching that of the serializer
pix2frm = visionhdl.PixelsToFrame(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines);

% Convert the pixel stream to an image frame
[frmOutput,frmValid] = step(pix2frm,pixOut,ctrlOut);
if frmValid
```

```
    figure
    imshow(frmOutput, 'InitialMagnification',300)
    title 'Output Image'
end
```





- “Accelerate a Pixel-Streaming Design Using MATLAB Coder”

Algorithm

This object implements the algorithms described on the `Gamma Corrector` block reference page.

See Also

vision.GammaCorrector | visionhdl.FrameToPixels | `Gamma Corrector` | `imadjust`

Introduced in R2015a

clone

System object: visionhdl.GammaCorrector
Package: visionhdl

Create object with same property values

Syntax

`newG = clone(G)`

Description

`newG = clone(G)` creates another instance of the `GammaCorrector` System object, `newG`, with the same property values as input argument `G`. The new object is unlocked and contains uninitialized states.

Input Arguments

`G`

`visionhdl.GammaCorrector` System object

Output Arguments

`newG`

New `GammaCorrector` System object with the same property values as the original System object.

Introduced in R2015a

isLocked

System object: visionhdl.GammaCorrector
Package: visionhdl

Locked status (logical)

Syntax

`TF = isLocked(G)`

Description

`TF = isLocked(G)` returns the locked status, `TF`, of the `GammaCorrector` System object, `G`.

Introduced in R2015a

release

System object: visionhdl.GammaCorrector
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

```
release(G)
```

Description

`release(G)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

G

`visionhdl.GammaCorrector` System object

Introduced in R2015a

step

System object: visionhdl.GammaCorrector
Package: visionhdl

Apply or remove gamma correction on one pixel

Syntax

`[pixelOut,ctrlOut] = step(G,pixelIn,ctrlIn)`

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[pixelOut,ctrlOut] = step(G,pixelIn,ctrlIn)` returns the intensity value of a pixel after gamma correction, and the control signals associated with the pixel. The input, `pixelIn`, and output, `pixelOut`, are scalar values representing a single pixel.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

G

visionhdl.GammaCorrector System object.

pixelIn

Intensity of a single pixel, specified as a scalar value. For fixed-point data types, the input word length must be less than or equal to 16.

Supported data types:

- int8 and int16
- uint8 and uint16
- fixdt()

double and single data types are supported for simulation but not for HDL code generation.

ctrlIn

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five logical signals. See "Pixel Control Structure".

Output Arguments

pixelOut

Gamma-corrected intensity of a single pixel, specified as a scalar value. The data type of the output pixel is the same as the data type of pixelIn.

ctrlOut

Control signals indicating the validity of the output pixel and the location of the pixel within the frame, returned as a structure containing five logical signals. See "Pixel Control Structure".

Introduced in R2015a

visionhdl.Histogram System object

Package: visionhdl

Frequency distribution

Description

`visionhdl.Histogram` computes the frequency distribution of pixel values in a video stream. You can configure the number and size of the bins. The object provides a read interface for accessing each bin. The object keeps a running histogram until you clear the bin values.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface allows object operation independent of image size and format, and easy connection with other Vision HDL Toolbox objects. The `step` method accepts pixel data as integer, fixed-point, or floating-point data types. The `step` method accepts control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = visionhdl.Histogram` returns a System object, `H`, that computes image histograms over 256 bins, with a bin size of 16 bits.

`H = visionhdl.Histogram(Name,Value)` returns a System object, `H`, with additional options specified by one or more `Name,Value` pair arguments. `Name` is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Properties not specified retain their default values.

Properties

`NumBins`

Number of bins for the histogram.

Choose the number of bins depending on the input word length (WL). If the number of bins is less than 2^{WL}, the object truncates the least-significant bits of each pixel. If the number of bins is greater than 2^{WL}, the object warns about an inefficient use of hardware resources.

Default: `256`

`OutputDataType`

Data type of the histogram values.

- `double`
- `single`
- `Unsigned fixed point` (default)

`double` and `single` data types are supported for simulation but not for HDL code generation.

`OutputWordLength`

Histogram bin value word length when `OutputDataType` is `Unsigned fixed point`. If a bin overflows, the count saturates and the object shows a warning.

Default: `16`

Methods

clone	Create object having the same property values
isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics

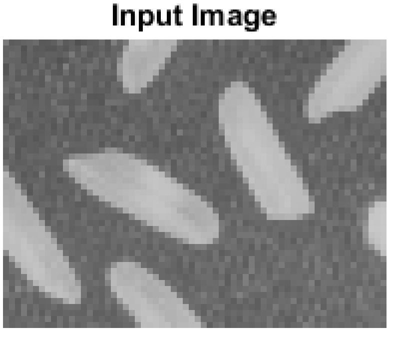
step	Sort input pixel into histogram bin, or read histogram bin

Examples

Compute Histogram of an Image

Set the dimensions of the test image, and load a source image. Select a portion of the image matching the desired test size.

```
frmActivePixels = 64;
frmActiveLines = 48;
frmOrig = imread('rice.png');
frmInput = frmOrig(1:frmActiveLines,1:frmActivePixels);
figure
imshow(frmInput,'InitialMagnification',300)
title 'Input Image'
```



Create a serializer object and define inactive pixel regions. Then, create a histogram object. The default setting is 256 bins.

```
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
```

```
        'ActiveVideoLines',frmActiveLines,...
        'TotalPixelsPerLine',frmActivePixels+10,...
        'TotalVideoLines',frmActiveLines+10,...
        'StartingActiveLine',6,...
        'FrontPorch',5);

histo = visionhdl.Histogram();
bins = str2double(histo.NumBins);
```

Serialize the test image. `pixIn` is a vector of intensity values and `ctrlIn` is a vector of control signal structures. Initialize output signals for the histogram results.

```
[pixIn,ctrlIn] = step(frm2pix,frmInput);

[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
readRdy = false(numPixelsPerFrame,1);
dataOut = zeros(bins-1,1,'uint8');
validOut  = false(bins-1,1);
noOpCtrl = pixelcontrolstruct(0,0,0,0,0);
noAddr = uint8(0);
noReset = false;
```

Call the object with dummy input to initialize the bin memory.

```
for p = 1:bins
    step(histo,uint8(0),noOpCtrl,noAddr,noReset);
end
```

For each pixel in the padded frame, sort the pixel into a bin. `readRdy` is returned true 2 cycles after the active frame is complete.

```
for p = 1:numPixelsPerFrame
   [~,readRdy(p),~] = step(histo,pixIn(p),ctrlIn(p),noAddr,noReset);
end
```

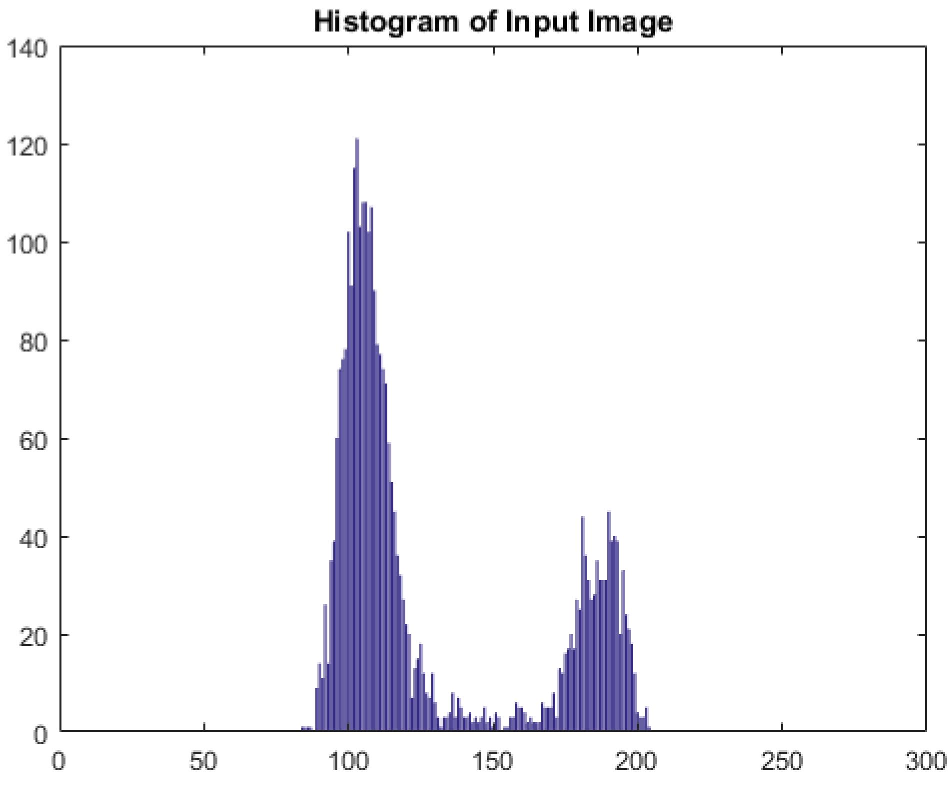
Once the frame is complete, as indicated by `readRdy`, read the bin values.

```
if readRdy(numPixelsPerFrame)
  for p = 1:bins+1
     if (p < bins-1)
        % Read a normal bin
        % Bin addresses are 0:bins-1
        [dataOut(p),~,validOut(p)] = step(histo,uint8(0),noOpCtrl,uint8(p-1),noReset);
     elseif (p == bins-1)
        % Read the final bin value and initiate binReset
```

```
        [dataOut(p),~,validOut(p)] = step(histo,uint8(0),noOpCtrl,uint8(bins-1),true);
     elseif (p >= bins)
        % Flush final bin values with 2 more calls to step
        [dataOut(p),~,validOut(p)] = step(histo,uint8(0),noOpCtrl,noAddr,noReset);
     end
  end
end
```

Graph the bin values.

```
dataOut = dataOut(validOut==1);
figure
bar(dataOut)
title('Histogram of Input Image')
```



Call the object with dummy input to clear the bin memory.

```
for p = 1:bins-2
    step(histo,uint8(0),noOpCtrl,noAddr,noReset);
end
```

Algorithm

This object implements the algorithms described on the `Histogram` block reference page.

See Also

`Histogram` | visionhdl.FrameToPixels | vision.Histogram | `imhist`

Introduced in R2015a

clone

System object: visionhdl.Histogram
Package: visionhdl

Create object having the same property values

Syntax

`newH = clone(H)`

Description

`newH = clone(H)` creates another instance of the `Histogram` System object, `H`, that has the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

`H`

`visionhdl.Histogram` System object

Output Arguments

`newH`

New `Histogram` System object that has the same property values as the original System object. The new unlocked object contains uninitialized states.

Introduced in R2015a

isLocked

System object: visionhdl.Histogram
Package: visionhdl

Locked status (logical)

Syntax

```
TF = isLocked(H)
```

Description

`TF = isLocked(H)` returns the locked status, `TF`, of the `Histogram` System object, `H`.

Introduced in R2015a

release

System object: visionhdl.Histogram
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

```
release(H)
```

Description

`release(H)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

H

`visionhdl.Histogram` System object

Introduced in R2015a

step

System object: visionhdl.Histogram
Package: visionhdl

Sort input pixel into histogram bin, or read histogram bin

Syntax

```
step(H,~,~,~,~)
[dataOut,readRdy,validOut] = step(H,pixelIn,ctrlIn,~,0)
[dataOut,readRdy,validOut] = step(H,~,~,binAddr,0)
[dataOut,readRdy,validOut] = step(H,~,~,binAddr,binReset)
```

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`step(H,~,~,~,~)` performs an initial reset phase before processing input data. After object creation or reset, call `step` with dummy arguments for `NumberOfBins` cycles before applying data. You do not have to assert `binReset` during this phase.

`[dataOut,readRdy,validOut] = step(H,pixelIn,ctrlIn,~,0)` adds the input pixel, `pixelIn`, to the internal histogram. Call `step` with this syntax for each pixel in a frame. The object returns `readRdy` `true` when the histogram for the frame is complete.

`[dataOut,readRdy,validOut] = step(H,~,~,binAddr,0)` reads the histogram bin specified by `binAddr`. Use this syntax when `readRdy` is returned `true`. Call `step` with this syntax for each histogram bin. The bin value at `binAddr` is returned in `dataOut`, with `validOut` set to `true`, after two further calls to `step`.

`[dataOut,readRdy,validOut] = step(H,~,~,binAddr,binReset)` resets the histogram values when `binReset` is `true`. You can initiate the reset while simultaneously giving the final binAddr. Before applying more video data, complete the reset sequence by calling step with dummy arguments for NumBins cycles.

To visualize the sequence of operations, see the timing diagrams in the “Algorithm” on page 1-76 section of the Histogram block page.

Note: The System object performs an initialization the first time you call the step method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the release method to unlock the object.

Input Arguments

H

visionhdl.Histogram System object.

pixelIn

Single pixel, specified by a scalar value.

Supported data types:

- uint
- fixdt(0,N,0)
- double and single data types are supported for simulation but not for HDL code generation.

ctrlIn

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five logical signals. See “Pixel Control Structure”.

binAddr

Bin number request for reading histogram values. This input is captured after readRdy is returned true. The data type is fixdt(0,N,0), N = 5,6,...,10. The word length must be $\log_2$(NumBins).

`binReset`

Triggers histogram RAM reset when true. Reset takes `NumBins` cycles to clear all locations. Input signals are ignored during this interval. Data type is `logical`.

Output Arguments

`readRdy`

Flag indicating when the histogram bins are ready for read, returned as a `logical` value. The object returns `readRdy` set to `true` two cycles after the final pixel of a frame.

`dataOut`

Histogram value for the bin requested in `binAddr`. The `OutputDataType` property specifies the data type for this output.

`validOut`

Flag indicating the validity of `dataOut`, returned as a `logical` value.

Introduced in R2015a

visionhdl.ImageFilter System object

Package: visionhdl

2-D FIR filtering

Description

`visionhdl.ImageFilter` performs two-dimensional FIR filtering on a pixel stream.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`F = visionhdl.ImageFilter` returns a System object, `F`, that performs two-dimensional FIR filtering on an input pixel stream.

`F = visionhdl.ImageFilter(Name,Value)` returns a 2-D FIR filter System object, `F`, with additional options specified by one or more `Name,Value` pair arguments. `Name` is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Properties not specified retain their default values.

`F = visionhdl.ImageFilter(coeff,lineSize,Name,Value)` returns a 2-D FIR filter System object, `F`, with the `Coefficients` property set to `coeff`, the `LineBufferSize` property to `lineSize`, and additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

coeff

Filter coefficients, specified as a vector or matrix. The maximum size along any dimension of this matrix or vector is 16. This argument sets the `Coefficients` property value.

lineSize

Size of the line memory buffer, specified as a power of two that accommodates the number of active pixels in a horizontal line. This argument sets the `LineBufferSize` property value.

Output Arguments

F

`visionhdl.ImageFilter` System object

Properties

Coefficients

Coefficients of the filter, specified as a vector or matrix of any numeric type. The maximum size along any dimension of this matrix or vector is 16.

`double` and `single` data types are supported for simulation but not for HDL code generation.

Default: `[1,0;0,-1]`

CoefficientsDataType

Method for determining the data type of the filter coefficients.

- `'Same as first input'` (default) — Sets the data type of the coefficients to match the data type of the `pixelIn` argument of the `step` method.
- `'custom'` — Sets the data type of the coefficients to match the data type defined in the `CustomCoefficientsDataType` property.

CustomCoefficientsDataType

Data type for the filter coefficients, specified as `numerictype(signed,WL,FL)`, where *WL* is the word length and *FL* is the fraction length in bits. This property applies when you set `CoefficientsDataType` to '`custom`'.

Default: `numerictype(true,16,15)`

CustomOutputDataType

Data type for the output pixels, specified as `numerictype(signed,WL,FL)`, where *WL* is the word length and *FL* is the fraction length in bits. This property applies only when you set `OutputDataType` to `custom`.

Default: `numerictype(true,8,0)`

LineBufferSize

Size of the line memory buffer, specified as a scalar integer.

Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the object uses the next largest power of two. The object allocates (*coefficient rows – 1*)-by-`LineBufferSize` memory locations to store the pixels.

Default: 2048

OutputDataType

Method for determining the data type of the output pixels.

- '`Same as first input`' (default) — Sets the data type of the output pixels to match the data type of the `pixelIn` argument of the `step` method.
- '`full precision`' — Computes internal and output data types using full precision rules. These rules provide accurate fixed-point numerics and prevent quantization within the object. Bits are added, as needed, to prevent rounding and overflow.
- '`custom`' — Sets the data type of the output pixels to match the data type you define in the `CustomOutputDataType` property.

OverflowAction

Overflow action used for fixed-point operations.

The object uses fixed-point arithmetic for internal calculations when the input is any integer or fixed-point data type. This option does not apply when the input data type is `single` or `double`.

Default: `Wrap`

`PaddingMethod`

Method for padding the boundary of the input image. See "Edge Padding".

- `'Constant'` (default) — Pads the input matrix with a constant value.
- `'Replicate'` — Repeats the value of pixels at the edge of the image.
- `'Symmetric'` — Pads the input matrix with its mirror image.

`PaddingValue`

Constant value used to pad the boundary of the input image. This property applies when you set `PaddingMethod` to `'Constant'`. The object casts this value to the same data type as the input pixel.

Default: 0

`RoundingMethod`

Rounding mode used for fixed-point operations.

The object uses fixed-point arithmetic for internal calculations when the input is any integer or fixed-point data type. This option does not apply when the input data type is `single` or `double`.

Default: `Floor`

Methods

clone	Create object with same property values
isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics

step	2-D FIR filtering

Examples

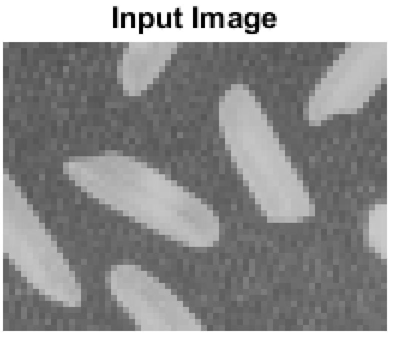
Filter a Pixel-Stream

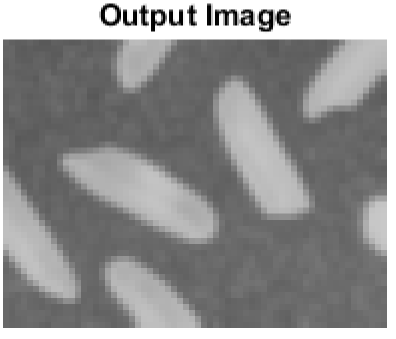
This example implements a 2-D blur filter on a thumbnail image.

```
% Set dimensions of the test image
frmActivePixels = 64;
frmActiveLines = 48;

% Load image source
frmOrig = imread('rice.png');

% Select a portion of the image matching the desired test size
frmInput = frmOrig(1:frmActiveLines,1:frmActivePixels);
figure
imshow(frmInput,'InitialMagnification',300)
title 'Input Image'

% Create serializer and define inactive pixel regions
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines,...
      'TotalPixelsPerLine',frmActivePixels+10,...
      'TotalVideoLines',frmActiveLines+10,...
      'StartingActiveLine',6,...
      'FrontPorch',5);

% Create filter object
 filt2d = visionhdl.ImageFilter(...
          'Coefficients',ones(2,2)/4,...
          'CoefficientsDataType','Custom',...
          'CustomCoefficientsDataType',numerictype(0,1,2),...
          'PaddingMethod','Symmetric');

% Serialize the test image
% pixIn is a vector of intensity values
% ctrlIn is a vector of control signal structures
```

```
[pixIn,ctrlIn] = step(frm2pix,frmInput);

% Prepare to process pixels
[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
pixOut = zeros(numPixelsPerFrame,1,'uint8');
ctrlOut  = repmat(pixelcontrolstruct,numPixelsPerFrame,1);

% For each pixel in the padded frame, compute the filtered value
% Monitor the control signals to determine latency of the object
% The latency of a filter configuration depends on:
% * the number of active pixels in a line
% * the size of the filter kernel
% * optimization of symmetric or duplicate coefficients
foundValIn = false;
foundValOut = false;
for p = 1:numPixelsPerFrame
    if (ctrlIn(p).valid && foundValIn==0)
        foundValIn = p;
    end
    [pixOut(p),ctrlOut(p)] = step(filt2d,pixIn(p),ctrlIn(p));
    if (ctrlOut(p).valid && foundValOut==0)
        foundValOut = p;
    end
end
sprintf('object latency is %d cycles',foundValOut-foundValIn)

% Create deserializer with format matching that of the serializer
pix2frm = visionhdl.PixelsToFrame(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines);

% Convert the pixel stream to an image frame
[frmOutput,frmValid] = step(pix2frm,pixOut,ctrlOut);
if frmValid
    figure
    imshow(frmOutput, 'InitialMagnification',300)
    title 'Output Image'
end
```

```
ans =

object latency is 101 cycles
```





Algorithm

This object implements the algorithms described on the `Image Filter` block reference page.

See Also

`Image Filter` | visionhdl.FrameToPixels | vision.ImageFilter | `imfilter`

Introduced in R2015a

clone

System object: visionhdl.ImageFilter
Package: visionhdl

Create object with same property values

Syntax

`newF = clone(F)`

Description

`newF = clone(F)` creates another instance of the `ImageFilter` System object, `F`, with the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

F

`visionhdl.ImageFilter` System object.

Output Arguments

newF

New `ImageFilter` System object with the same property values as the original System object.

Introduced in R2015a

isLocked

System object: visionhdl.ImageFilter
Package: visionhdl

Locked status (logical)

Syntax

```
TF = isLocked(F)
```

Description

`TF = isLocked(F)` returns the locked status, `TF`, of the `ImageFilter` System object, `F`.

Introduced in R2015a

release

System object: visionhdl.ImageFilter
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

`release(F)`

Description

`release(F)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Note: You can use the release method on a System object in code generated from MATLAB®, but once you release its resources, you cannot use that System object again.

Input Arguments

F

`visionhdl.ImageFilter` System object

Introduced in R2015a

step

System object: visionhdl.ImageFilter
Package: visionhdl

2-D FIR filtering

Syntax

`[pixelOut,ctrlOut] = step(F,pixelIn,ctrlIn)`

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[pixelOut,ctrlOut] = step(F,pixelIn,ctrlIn)` returns the next pixel, `pixelOut`, of the filtered image resulting from applying the coefficients in the `Coefficients` property to the image described by the input pixel stream, `pixelIn`.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

F

`visionhdl.ImageFilter` System object.

`pixelIn`

Single pixel, specified as a scalar value.

Supported data types:

- `uint` or `int`
- `fixdt()`
- `double` and `single` data types are supported for simulation but not for HDL code generation.

`ctrlIn`

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five `logical` signals. See “Pixel Control Structure”.

Output Arguments

`pixelOut`

Single filtered pixel, returned as a scalar value.

Configure the data type of the output pixel by using the `OutputDataType` and `CustomOutputDataType` properties.

`ctrlOut`

Control signals indicating the validity of the output pixel and the location of the pixel within the frame, returned as a structure containing five `logical` signals. See “Pixel Control Structure”.

Introduced in R2015a

visionhdl.ImageStatistics System object

Package: visionhdl

Mean, variance, and standard deviation

Description

`visionhdl.ImageStatistics` calculates the mean, variance, and standard deviation of streaming video data. Each calculation is performed over all pixels in the input region of interest (ROI). The object implements the calculations using hardware-efficient algorithms.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface allows object operation independent of image size and format, and easy connection with other Vision HDL Toolbox objects. The `step` method accepts pixel data as integer, fixed-point, or floating-point data types. The `step` method accepts control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

- To change the size and dimensions of the ROI, you can manipulate the input video stream control signals. See “Regions of Interest” on page 1-108.
- The number of valid pixels in the input image affect the accuracy of the mean approximation. To avoid approximation error, use an image that contains fewer than 64 pixels, a multiple of 64 pixels up to 64^2 pixels, a multiple of 4096 pixels up to 64^3 pixels, or a multiple of 64^3 pixels up to 64^4 pixels. For details of the mean approximation, see “Algorithm” on page 1-103.
- The object calculates statistics over frames up to 64^4 (16,777,216) pixels in size.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

S = visionhdl.ImageStatistics returns a System object, S, that calculates the mean, variance, and standard deviation of each frame of a video stream.

S = visionhdl.ImageStatistics(Name,Value) returns a System object, S, with additional options specified by one or more Name,Value pair arguments. Name is a property name and Value is the corresponding value. Name must appear inside single quotes (''). You can specify several name-value pair arguments in any order as Name1,Value1,...,NameN,ValueN. Properties not specified retain their default values.

Properties

mean

Calculate the mean of each input frame. If you set this property to false, the step method does not return this output.

Default: true

variance

Calculate the variance of each input frame. If you set this property to false, the step method does not return this output.

Default: true

stdDev

Calculate the standard deviation of each input frame. If you set this property to false, the step method does not return this output.

Default: true

Methods

clone	Create object having the same property values

isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics
step	Calculate the contribution of one pixel to the mean, variance, and standard deviation of a video stream

Examples

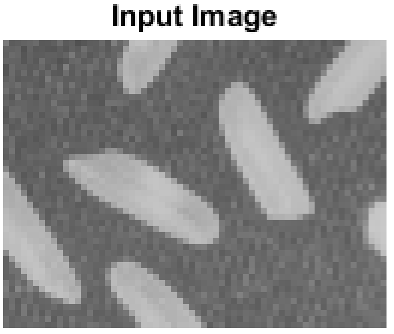
Compute Statistics of an Image

This example computes the mean, variance, and standard deviation of a thumbnail image.

```
% Set dimensions of the test image
frmActivePixels = 64;
frmActiveLines = 48;

% Load image source
frmOrig = imread('rice.png');

% Select a portion of the image matching the desired test size
frmInput = frmOrig(1:frmActiveLines,1:frmActivePixels);
figure
imshow(frmInput,'InitialMagnification',300)
title 'Input Image'

% Create serializer and define inactive pixel regions
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines,...
      'TotalPixelsPerLine',frmActivePixels+10,...
      'TotalVideoLines',frmActiveLines+10,...
      'StartingActiveLine',6,...
      'FrontPorch',5);

 % Create object that returns mean, variance, and standard deviation
 stats = visionhdl.ImageStatistics();
```

```
% Serialize the test image
% pixIn is a vector of intensity values
% ctrlIn is a vector of control signal structures
[pixIn,ctrlIn] = step(frm2pix,frmInput);

% Prepare to process pixels
[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
validOut  = false(numPixelsPerFrame,1);
mean  = zeros(numPixelsPerFrame,1,'uint8');
variance  = zeros(numPixelsPerFrame,1,'uint8');
stddev  = zeros(numPixelsPerFrame,1,'uint8');

% For each pixel in the stream, increment the internal statistics
for p = 1:numPixelsPerFrame
   [mean(p),variance(p),stddev(p),validOut(p)] = step(stats,pixIn(p),ctrlIn(p));
end

% The results are valid when validOut is returned true
mean = mean(validOut==1)
variance = variance(validOut==1)
stddev = stddev(validOut==1)
```

```
mean =

  uint8

   125


variance =

  uint8

   255


stddev =

  uint8

   36
```



Algorithm

This object implements the algorithms described on the `Image Statistics` block reference page.

See Also

`Image Statistics` | vision.Variance | visionhdl.FrameToPixels | vision.Mean | vision.StandardDeviation | `mean2` | `std2`

Introduced in R2015a

clone

System object: visionhdl.ImageStatistics
Package: visionhdl

Create object having the same property values

Syntax

`newS = clone(S)`

Description

`newS = clone(S)` creates another instance of the `ImageStatistics` System object, `S`, that has the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

S

`visionhdl.ImageStatistics` System object

Output Arguments

newS

New `ImageStatistics` System object that has the same property values as the original System object.

Introduced in R2015a

isLocked

System object: visionhdl.ImageStatistics
Package: visionhdl

Locked status (logical)

Syntax

```
TF = isLocked(S)
```

Description

`TF = isLocked(S)` returns the locked status, `TF`, of the `ImageStatistics` System object, `S`.

Introduced in R2015a

release

System object: visionhdl.ImageStatistics
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

`release(S)`

Description

`release(S)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

S

`visionhdl.ImageStatistics`System object

Introduced in R2015a

step

System object: visionhdl.ImageStatistics
Package: visionhdl

Calculate the contribution of one pixel to the mean, variance, and standard deviation of a video stream

Syntax

`[mean,variance,stdDeviation,validOut] = step(S,pixelIn,ctrlIn)`

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[mean,variance,stdDeviation,validOut] = step(S,pixelIn,ctrlIn)` incorporates the new pixel value, `pixelIn`, into calculations of video frame statistics. The control signals associated with each pixel, `ctrlIn`, indicate the frame boundaries. When `validOut` is `true`, the output values of `mean`, `variance`, and `stdDeviation` represent the statistics for the most recent input frame completed. The number of statistics returned depends on the object property settings.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface allows object operation independent of image size and format, and easy connection with other Vision HDL Toolbox objects. The `step` method accepts pixel data as integer, fixed-point, or floating-point data types. The `step` method accepts control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

S

`visionhdl.ImageStatistics` System object.

pixelIn

Single pixel, specified as a scalar value.

Supported data types:

- `uint8` or `uint16`
- `fixdt(0,N,0)`, N = 8,9,...,16
- `double` and `single` data types are supported for simulation but not for HDL code generation.

ctrlIn

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five `logical` signals. See “Pixel Control Structure”.

Output Arguments

mean

Mean of the most recent frame of video input, returned as a scalar value.

The data type is the same as `pixelIn`.

variance

Variance of the most recent frame of video input, returned as a scalar value.

The data type is the same as `pixelIn`. The fixed-point output word length is double the input word length.

`stdDeviation`

Standard deviation of the most recent frame of video input, returned as a scalar value.

The data type is the same as `pixelIn`. Fixed-point output word length is double the input word length.

`validOut`

Validity of output statistics. When the object completes the calculations, it returns `true`. When this output is `true`, the other output arguments are valid. Data type is `logical`.

Introduced in R2015a

visionhdl.LookupTable System object

Package: visionhdl

Map input pixel to output pixel using custom rule

Description

The `visionhdl.LookupTable` System object uses a custom one-to-one map to convert between an input pixel value and an output pixel value.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`LUT = visionhdl.LookupTable` returns a System object, `LUT`, that performs a one-to-one mapping between the input pixel and output pixel, according to the lookup table contents.

`LUT = visionhdl.LookupTable(tabledata)` returns a lookup table System object, `LUT`, with the table contents set to `TABLEDATA`.

Input Arguments

`tabledata`

One-to-one correspondence between input pixels and output pixels, specified as a vector. This argument sets the `Table` property value.

Output Arguments

`LUT`

`visionhdl.visionhdl.LookupTable` System object

Properties

`Table`

Map between input pixel values and output pixel values.

- The table data is a vector, row or column, of any data type. The data type of the table data determines that of `pixelOut`. See `visionhdl.LookupTable.step` method.
- The length of the table data must equal $2^{WordLength}$, where *WordLength* is the size, in bits, of `pixelIn`. See `visionhdl.LookupTable.step` method.
- The smallest representable value of the input data type maps to the first element of the table, the second smallest value maps to the second element, and so on. For example, if `pixelIn` has a data type of `fixdt(0,3,1)`, the input value 0 maps to the first element of the table, input value 0.5 maps to the second element, 1 maps to the third, and so on.

Default: `uint8(0:1:255)`

Methods

clone	Create object with same property values
isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics
step	Map input pixel to output pixel based on table contents

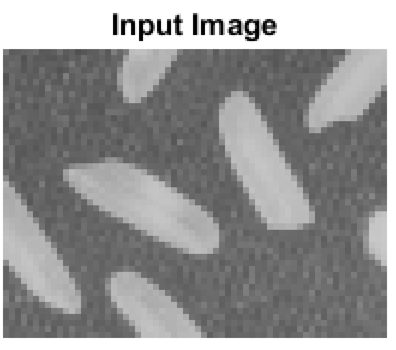
Examples

Compute Negative Image

This example creates the negative of an image by looking up the opposite pixel values in a table.

Set dimensions of test image, and load an image source. Select a portion of the image matching the desired test size.

```
frmActivePixels = 64;
frmActiveLines = 48;
frmOrig = imread('rice.png');
frmInput = frmOrig(1:frmActiveLines,1:frmActivePixels);
figure
imshow(frmInput,'InitialMagnification',300)
title 'Input Image'
```



Create a serializer object and define inactive pixel regions.

```
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
```

```
      'ActiveVideoLines',frmActiveLines,...
      'TotalPixelsPerLine',frmActivePixels+10,...
      'TotalVideoLines',frmActiveLines+10,...
      'StartingActiveLine',6,...
      'FrontPorch',5);
```

Create a lookup table object. The input pixel data is uint8 type, so the negative value is 255-|pixel|. The output pixel data type is the same as the data type of the table contents.

```
tabledata = uint8(linspace(255,0,256));
inverter = visionhdl.LookupTable(tabledata);
```

Serialize the test image. `pixIn` is a vector of intensity values. `ctrlIn` is a vector of control signal structures.

```
[pixIn,ctrlIn] = step(frm2pix,frmInput);
```

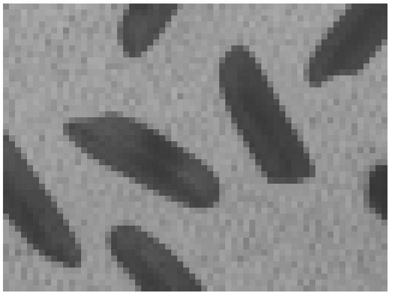
Prepare variables to process pixels. Then, for each pixel in the padded frame, look up the negative value.

```
[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
pixOut = zeros(numPixelsPerFrame,1,'uint8');
ctrlOut  = repmat(pixelcontrolstruct,numPixelsPerFrame,1);
for p = 1:numPixelsPerFrame
    [pixOut(p),ctrlOut(p)] = step(inverter,pixIn(p),ctrlIn(p));
end
```

Create deserializer object with a video format matching that of the serializer. Convert the output pixel stream to an image frame, and display the result.

```
pix2frm = visionhdl.PixelsToFrame(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines);
[frmOutput,frmValid] = step(pix2frm,pixOut,ctrlOut);
if frmValid
    figure
    imshow(frmOutput, 'InitialMagnification',300)
    title 'Output Image'
end
```

Output Image



Algorithm

This object implements the algorithms described on the `Lookup Table` block reference page.

See Also

`Lookup Table` | visionhdl.FrameToPixels

Introduced in R2015a

clone

System object: visionhdl.LookupTable
Package: visionhdl

Create object with same property values

Syntax

```
newLUT = clone(LUT)
```

Description

`newLUT = clone(LUT)` creates another instance of the `LookupTable` System object, `newLUT`, with the same property values as input argument `LUT`. The new object is unlocked and contains uninitialized states.

Input Arguments

`LUT`

`visionhdl.LookupTable` System object

Output Arguments

`newLUT`

New `LookupTable` System object with the same property values as the original System object.

Introduced in R2015a

isLocked

System object: visionhdl.LookupTable
Package: visionhdl

Locked status (logical)

Syntax

```
TF = isLocked(LUT)
```

Description

`TF = isLocked(LUT)` returns the locked status, `TF`, of the `LookupTable` System object, `LUT`.

Introduced in R2015a

release

System object: visionhdl.LookupTable
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

`release(LUT)`

Description

`release(LUT)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

LUT

`visionhdl.LookupTable` System object

Introduced in R2015a

step

System object: visionhdl.LookupTable
Package: visionhdl

Map input pixel to output pixel based on table contents

Syntax

```
[pixelOut,ctrlOut] = step(LUT,pixelIn,ctrlIn)
```

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[pixelOut,ctrlOut] = step(LUT,pixelIn,ctrlIn)` returns the pixel value, `pixelOut`, located in the table at the address specified by the input pixel value, `pixelIn`. The object passes the control signals, `ctrlIn`, through and aligns the output control signals, `ctrlOut`, with the output data.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

LUT

visionhdl.LookupTable System object

pixelIn

Input pixel, specified as a scalar value. For unsigned fixed-point data types, the input word length must be less than or equal to 16.

Supported data types:

- logical
- uint8 or uint16
- fixdt()

ctrlIn

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five logical signals. See "Pixel Control Structure".

Output Arguments

pixelOut

Output pixel, returned as a scalar value. The data type of the output is the same as the data type of the entries you specify in the Table property.

ctrlOut

Control signals indicating the validity of the output pixel and the location of the pixel within the frame, returned as a structure containing five logical signals. See "Pixel Control Structure".

Introduced in R2015a

visionhdl.MeasureTiming System object

Package: visionhdl

Measure timing of pixel control structure input

Description

The `visionhdl.MeasureTiming` object measures the timing parameters of a video stream. The Vision HDL Toolbox streaming pixel protocol implements the timing of a video system, including inactive intervals between frames. These inactive intervals are called *blanking intervals*. Many Vision HDL Toolbox objects require minimum blanking intervals. You can use the timing parameter measurements from this object to check that your video stream meets these requirements. If you manipulate the control signals of your video stream, you can use this object to verify the resulting control signals. To determine the parameters of each frame, the object measures time steps between the control signals in the input structure.

For details on the pixel control bus and the dimensions of a video frame, see "Streaming Pixel Interface".

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`measure = visionhdl.MeasureTiming` returns a System object, `measure`, that measures the average frame timing of a video stream.

Methods

clone	Create object having the same property values
isLocked	Locked status (logical)

release	Allow changes to property values and input characteristics
step	Measure timing of pixel control structure input

Examples

Measure Timing Parameters of Custom Video Stream

This example shows how to use the `MeasureTiming` object to observe the frame parameters in a custom video stream. The example creates customized padding around an image frame and converts the frame to streaming video. It uses the `MeasureTiming` object to confirm that the streaming video parameters match the custom settings.

Use a `FrameToPixels` object to specify a small custom-size frame with customized blanking intervals. To obtain a frame of this size, select a small section of the input image.

```
frm2pix = visionhdl.FrameToPixels(...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',32,...
      'ActiveVideoLines',18,...
      'TotalPixelsPerLine',42,...
      'TotalVideoLines',26,...
      'StartingActiveLine',6,...
      'FrontPorch',5);
[actPixPerLine,actLine,numPixPerFrm] = getparamfromfrm2pix(frm2pix);

frmFull = imread('rice.png');
frmIn = frmFull(74:73+actLine,104:103+actPixPerLine);
imshow(frmIn);
```

Create a `MeasureTiming` object to observe the parameters of the serial pixel output from the `FrameToPixels` object.

```
measure = visionhdl.MeasureTiming;
```

Serialize the input frame.

```
[pixInVec,ctrlInVec] = frm2pix(frmIn);
```

Some parameters require measurements between frames, so you must simulate at least two frames before using the results. Because you serialized only one input frame, process that frame twice to measure all parameters correctly.

```
for f = 1:2
      for p = 1:numPixPerFrm
          [activePixels,activeLines,totalPixels,totalLines,...
              horizBlank,vertBlank] = measure(ctrlInVec(p));
      end
      fprintf('\nFrame %d:\n', f);
      fprintf('activePixels: %f\n', activePixels);
      fprintf('activeLines: %f\n', activeLines);
      fprintf('totalPixels: %f\n', totalPixels);
      fprintf('totalLines: %f\n', totalLines);
      fprintf('horizBlank: %f\n', horizBlank);
      fprintf('vertBlank: %f\n', vertBlank);
end
```

```
Frame 1:
activePixels: 32.000000
activeLines: 18.000000
totalPixels: 42.000000
totalLines: 22.880952
horizBlank: 10.000000
vertBlank: 4.880952

Frame 2:
activePixels: 32.000000
activeLines: 18.000000
totalPixels: 42.000000
totalLines: 26.000000
horizBlank: 10.000000
vertBlank: 8.000000
```

The measurements after the first frame are not accurate. However, after the second frame, the measurements match the parameters chosen in the `FrameToPixels` object.

See Also

`Measure Timing` | visionhdl.FrameToPixels

Introduced in R2016b

clone

System object: visionhdl.MeasureTiming
Package: visionhdl

Create object having the same property values

Syntax

`newM = clone(M)`

Description

`newM = clone(M)` creates another instance of the `ImageStatistics` System object, `M`, that has the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

`S`

`visionhdl.MeasureTiming` System object

Output Arguments

`newM`

New `MeasureTiming` System object that has the same property values as the original System object.

Introduced in R2016b

isLocked

System object: visionhdl.MeasureTiming
Package: visionhdl

Locked status (logical)

Syntax

`TF = isLocked(M)`

Description

`TF = isLocked(M)` returns the locked status, `TF`, of the `MeasureTiming` System object, `M`.

Introduced in R2016b

release

System object: visionhdl.MeasureTiming
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

`release(M)`

Description

`release(M)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

M

`visionhdl.MeasureTiming`System object

Introduced in R2016b

step

System object: visionhdl.MeasureTiming
Package: visionhdl

Measure timing of pixel control structure input

Syntax

```
[activePixels,activeLines,totalPixels,totalLines,horizBlank,vertBlank]
= step(measure,ctrlIn)
```

Description

Note: Alternatively, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[activePixels,activeLines,totalPixels,totalLines,horizBlank,vertBlank] = step(measure,ctrlIn)` incorporates the current pixel location into calculations of video frame timing. The control signals, `ctrlIn`, associated with each pixel indicate the location of this pixel relative to the active frame boundaries. The input object measures the time steps between the control signals to determine the parameters of each frame.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

`measure`

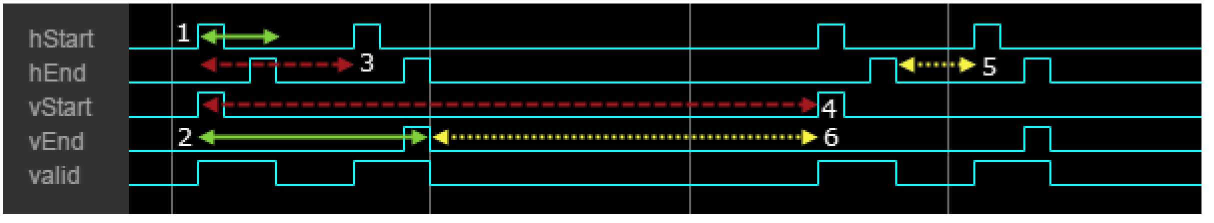
`visionhdl.MeasureTiming` System object.

`ctrlIn`

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five `logical` signals. See “Pixel Control Structure”.

Output Arguments

The diagram shows the output measurements, as determined from the pixel stream control signals.



For details on the pixel control bus and the dimensions of a video frame, see “Streaming Pixel Interface”.

Note: Measurements from the first simulated frame are incorrect because some parameters require measurements between frames. Simulate at least two frames before using the results.

`activePixels`

Number of active pixels per line.

This value is measured between `hStart` and `hEnd`. See marker `1` in the diagram.

activeLines

Number of active lines in the frame

This value is measured as the number of hStart pulses between vStart and vEnd. See marker 2 in the diagram.

totalPixels

Number of pixels in the line, including the horizontal blanking interval.

This value is measured from hStart to the next hStart. See marker 3 in the diagram.

totalLines

Number of lines in the frame, including the vertical blanking interval.

This value is measured by the interval from vEnd to the next vEnd, divided by totalPixels. See marker 4 in the diagram.

horizBlank

Number of pixels in the horizontal blanking interval.

The horizontal blanking interval is the number of inactive pixels between lines of a frame. This value is measured between hEnd and the next hStart. See marker 5 in the diagram.

vertBlank

Number of lines in the vertical blanking interval.

The vertical blanking interval is the number of inactive lines between frames. This value is measured from vEnd to the next vStart, adjusted to remove horizBlank, then divided by totalPixels. See marker 6 in the diagram.

Introduced in R2016b

visionhdl.MedianFilter System object

Package: visionhdl

2-D median filtering

Description

`visionhdl.MedianFilter` performs 2-D median filtering on a pixel stream. The object replaces each pixel value with the median value of the adjacent pixels.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`MF = visionhdl.MedianFilter` returns a System object, `MF`, that performs two-dimensional median filtering of serial pixel data.

`MF = visionhdl.MedianFilter(Name,Value)` returns a median filter System object, `MF`, with additional options specified by one or more `Name`,`Value` pair arguments. `Name` is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Properties not specified retain their default values.

`MF = visionhdl.MedianFilter(size,Name,Value)` returns a median filter System object, `MF`, with the `NeighborhoodSize` property set to `size` and additional options specified by one or more `Name`,`Value` pair arguments.

Input Arguments

`size`

Size in pixels of the image region used to compute the median. This argument sets the `NeighborhoodSize` property value.

Output Arguments

`MF`

`visionhdl.MedianFilter` System object.

Properties

`NeighborhoodSize`

Neighborhood size, in pixels.

- '3×3' (default)
- '5×5'
- '7×7'

`LineBufferSize`

Size of the line memory buffer, specified as a scalar integer.

Choose a power of 2 that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the object uses the next largest power of two. The object allocates *N - 1*-by-`LineBufferSize` memory locations to store the pixels used to compute the median value. *N* is the number of lines in the square region specified in **Neighborhood size**.

Default: 2048

`PaddingMethod`

Method for padding the boundary of the input image

- '`Constant`' — Pad input matrix with a constant value.
- '`Replicate`' — Repeat the value of pixels at the edge of the image.

- 'Symmetric' (default) — Pad image edge with its mirror image.

PaddingValue

Constant value used to pad the boundary of the input image. This property applies when you set PaddingMethod to 'Constant'. The object casts this value to the same data type as the input pixel.

Default: 0

Methods

clone	Create object with same property values
isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics
step	Median pixel value of neighborhood

Examples

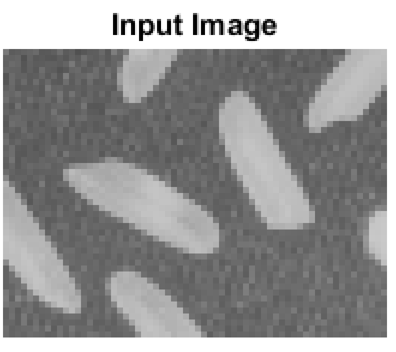
Median Filter on a Pixel-Stream

This example implements a 5×5 median filter on a thumbnail image.

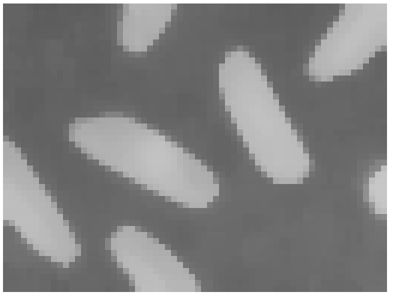
```
% Set the dimensions of the test image
frmActivePixels = 64;
frmActiveLines = 48;

% Load image source
frmOrig = imread('rice.png');

% Select a portion of the image matching the desired test size
frmInput = frmOrig(1:frmActiveLines,1:frmActivePixels);
figure
imshow(frmInput,'InitialMagnification',300)
title 'Input Image'

% Create serializer and define inactive pixel regions
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',1,...
```

```
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines,...
      'TotalPixelsPerLine',frmActivePixels+10,...
      'TotalVideoLines',frmActiveLines+10,...
      'StartingActiveLine',6,...
      'FrontPorch',5);

% Create filter
 mfilt = visionhdl.MedianFilter(...
          'NeighborhoodSize','5x5');

% Serialize the test image
% pixIn is a vector of intensity values
% ctrlIn is a vector of control signal structures
[pixIn,ctrlIn] = step(frm2pix,frmInput);

% Prepare to process pixels
[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
pixOut = zeros(numPixelsPerFrame,1,'uint8');
ctrlOut  = repmat(pixelcontrolstruct,numPixelsPerFrame,1);

% For each pixel in the padded frame, compute the local median
% Monitor control signals to determine latency of the object
% The latency of a filter configuration depends on:
% * the number of active pixels in a line
% * the size of the neighbourhood
foundValIn = false;
foundValOut = false;
for p = 1:numPixelsPerFrame
    if (ctrlIn(p).valid && foundValIn==0)
        foundValIn = p;
    end
    [pixOut(p),ctrlOut(p)] = step(mfilt,pixIn(p),ctrlIn(p));
    if (ctrlOut(p).valid && foundValOut==0)
        foundValOut = p;
    end
end
sprintf('object latency is %d cycles',foundValOut-foundValIn)

% Create deserializer with format matching that of the serializer
pix2frm = visionhdl.PixelsToFrame(...
      'NumComponents',1,...
      'VideoFormat','custom',...
```

```
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines);

% Convert the pixel stream to an image frame
[frmOutput,frmValid] = step(pix2frm,pixOut,ctrlOut);
if frmValid
    figure
    imshow(frmOutput, 'InitialMagnification',300)
    title 'Output Image'
end


ans =

object latency is 177 cycles
```



Output Image



Algorithm

This object implements the algorithms described on the `Median Filter` block reference page.

See Also

`Median Filter` | visionhdl.FrameToPixels | `medfilt2`

Introduced in R2015a

clone

System object: visionhdl.MedianFilter
Package: visionhdl

Create object with same property values

Syntax

`newF = clone(F)`

Description

`newF = clone(F)` creates another instance of the `MedianFilter` System object, `F`, with the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

F

`visionhdl.MedianFilter` System object

Output Arguments

newF

New `MedianFilter` System object with the same property values as the original System object.

Introduced in R2015a

isLocked

System object: visionhdl.MedianFilter
Package: visionhdl

Locked status (logical)

Syntax

`TF = isLocked(F)`

Description

`TF = isLocked(F)` returns the locked status, `TF`, of the `MedianFilter` System object, `F`.

Introduced in R2015a

release

System object: visionhdl.MedianFilter
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

`release(F)`

Description

`release(F)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

F

`visionhdl.MedianFilter` System object

Introduced in R2015a

step

System object: visionhdl.MedianFilter
Package: visionhdl

Median pixel value of neighborhood

Syntax

`[pixelOut,ctrlOut] = step(F,pixelIn,ctrlIn)`

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[pixelOut,ctrlOut] = step(F,pixelIn,ctrlIn)` returns the next pixel value, `pixelOut`, in the filtered pixel stream resulting from calculating the median of the neighborhood around each input pixel, `pixelIn`. Before filtering, the object pads image edges according to the `PaddingMethod` property.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

F

`visionhdl.MedianFilter` System object.

pixelIn

Single pixel, specified as a scalar value.

Supported data types:

- `uint` or `int`
- `fixdt(~,N,0)`
- `logical`
- `double` and `single` data types are supported for simulation but not for HDL code generation.

ctrlIn

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five `logical` signals. See “Pixel Control Structure”.

Output Arguments

pixelOut

Single pixel value representing the median of its neighborhood, returned as a scalar value.

The data type is the same as the data type of `pixelIn`.

ctrlOut

Control signals indicating the validity of the output pixel and the location of the pixel within the frame, returned as a structure containing five `logical` signals. See “Pixel Control Structure”.

Introduced in R2015a

visionhdl.Opening System object

Package: visionhdl

Morphological opening of binary pixel data

Description

`visionhdl.Opening` performs morphological erosion, followed by morphological dilation, using the same neighborhood for both calculations. The object operates on a stream of binary intensity values.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`O = visionhdl.Opening` returns a System object, `O`, that performs morphological opening on a binary pixel stream.

`O = visionhdl.Opening(Name,Value)` returns a System object, `O`, with additional options specified by one or more `Name,Value` pair arguments. `Name` is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Properties not specified retain their default values.

Properties

`Neighborhood`

Pixel neighborhood, specified as a matrix or vector of ones and zeros.

The object supports neighborhoods of up to 32×32 pixels. To use a structuring element, specify `Neighborhood` as `getnhood(strel(`*`shape`*`))`.

Default: `ones(3,3)`

`LineBufferSize`

Size of the line memory buffer, specified as a scalar integer.

Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the object uses the next largest power of two. The object allocates (*neighborhood lines* – 1)-by-`LineBufferSize` memory locations to store the pixels.

Default: `2048`

Methods

clone	Create object having the same property values
isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics
step	Report opened pixel value based on neighborhood

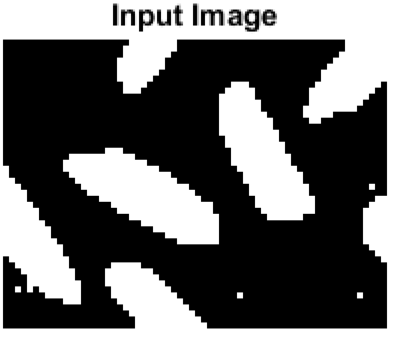
Examples

Morphological Open

Perform morphological open on a thumbnail image.

Load a source image from a file. Select a portion of the image that matches the desired test size. This source image contains `uint8` pixel intensity values. Apply a threshold to convert to binary pixel data.

```
frmOrig = imread('rice.png');
frmActivePixels = 64;
frmActiveLines = 48;
frmInput = frmOrig(1:frmActiveLines,1:frmActivePixels);
frmInput = frmInput>128;
figure
imshow(frmInput,'InitialMagnification',300)
title 'Input Image'
```



Create a serializer object and define inactive pixel regions. Make the number of inactive pixels following each active line at least double the horizontal size of the neighborhood. Make the number of lines following each frame at least double the vertical size of the neighborhood.

```
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines,...
      'TotalPixelsPerLine',frmActivePixels+20,...
      'TotalVideoLines',frmActiveLines+10,...
      'StartingActiveLine',3,...
```

```
      'FrontPorch',10);
```

Create a filter object.

```
 mopen = visionhdl.Opening( ...
          'Neighborhood',ones(5,5));
```

Serialize the test image by calling `step` on the serializer object. `pixIn` is a vector of intensity values. `ctrlIn` is a vector of control signal structures.

```
[pixIn,ctrlIn] = step(frm2pix,frmInput);
```

Prepare to process pixels by preallocating output vectors.

```
[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
pixOut = false(numPixelsPerFrame,1);
ctrlOut  = repmat(pixelcontrolstruct,numPixelsPerFrame,1);
```

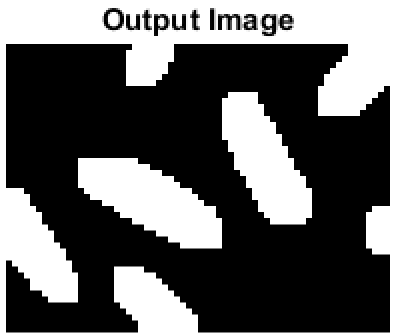
For each pixel in the padded frame, compute the morphed value. Monitor the control signals to determine latency of the object. The latency of a configuration depends on the number of active pixels in a line and the size of the neighborhood

```
foundValIn = false;
foundValOut = false;
for p = 1:numPixelsPerFrame
    if (ctrlIn(p).valid && foundValIn==0)
        foundValIn = p;
    end
    [pixOut(p),ctrlOut(p)] = step(mopen,pixIn(p),ctrlIn(p));
    if (ctrlOut(p).valid && foundValOut==0)
        foundValOut = p;
    end
end
sprintf('object latency is %d cycles',foundValOut-foundValIn)
```

```
ans =

object latency is 372 cycles
```

Create a deserializer object with a format matching that of the serializer. Convert the pixel stream to an image frame by calling `step` on the deserializer object. Display the resulting image.

```
pix2frm = visionhdl.PixelsToFrame(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines);
[frmOutput,frmValid] = step(pix2frm,pixOut,ctrlOut);
if frmValid
    figure
    imshow(frmOutput, 'InitialMagnification',300)
    title 'Output Image'
end
```



Algorithm

This object implements the algorithms described on the `Opening` block reference page.

See Also

visionhdl.Dilation | vision.MorphologicalOpen | visionhdl.Erosion | visionhdl.Closing | `Opening` | `imopen` | visionhdl.FrameToPixels

Introduced in R2015a

clone

System object: visionhdl.Opening
Package: visionhdl

Create object having the same property values

Syntax

```
newH = clone(O)
```

Description

`newH = clone(O)` creates another instance of the `Opening` System object, `O`, that has the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

O

`visionhdl.Opening` System object

Output Arguments

newO

New `Opening` System object that has the same property values as the original System object. The new unlocked object contains uninitialized states.

Introduced in R2015a

isLocked

System object: visionhdl.Opening
Package: visionhdl

Locked status (logical)

Syntax

```
TF = isLocked(O)
```

Description

`TF = isLocked(O)` returns the locked status, `TF`, of the `DemosiacInterpolator` System object, `O`.

Introduced in R2015a

release

System object: visionhdl.Opening
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

`release(O)`

Description

`release(O)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

O

`visionhdl.Opening` System object

Introduced in R2015a

step

System object: visionhdl.Opening
Package: visionhdl

Report opened pixel value based on neighborhood

Syntax

`[pixelOut,ctrlOut] = step(O,pixelIn,ctrlIn)`

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[pixelOut,ctrlOut] = step(O,pixelIn,ctrlIn)` returns the next pixel value, `pixelOut`, resulting from a morphological open operation on the neighborhood around each input pixel, `pixelIn`.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

O

`visionhdl.Opening` System object

pixelIn

Single pixel, specified as a scalar `logical` value.

ctrlIn

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five `logical` signals. See “Pixel Control Structure”.

Output Arguments

pixelOut

Single pixel transformed by a morphological operation, returned as a scalar `logical` value.

ctrlOut

Control signals indicating the validity of the output pixel and the location of the pixel within the frame, returned as a structure containing five `logical` signals. See “Pixel Control Structure”.

Introduced in R2015a

visionhdl.GrayscaleOpening System object

Package: visionhdl

Morphological opening of grayscale pixel data

Description

`visionhdl.GrayscaleOpening` performs morphological erosion, followed by morphological dilation, using the same neighborhood for both calculations. The object operates on a stream of pixel intensity values. You can specify a neighborhood, or structuring element, of up to 32×32 pixels. For line, square, or rectangle structuring elements more than 8 pixels wide, the object uses the Van Herk algorithm to find the maximum and minimum. For structuring elements less than 8 pixels wide, or that contain zero elements, the object implements a pipelined comparison tree to find the maximum and minimum.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`O = visionhdl.GrayscaleOpening` returns a System object, `O`, that performs morphological opening on a pixel stream.

`O = visionhdl.GrayscaleOpening(Name,Value)` returns a System object, `O`, with additional options specified by one or more `Name`,`Value` pair arguments. `Name`

is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Properties not specified retain their default values.

Properties

`Neighborhood`

Pixel neighborhood, specified as a matrix or vector of ones and zeros.

The object supports neighborhoods of up to 32×32 pixels. To use a structuring element, specify `Neighborhood` as `getnhood(strel(`*`shape`*`))`. The minimum neighborhood size is a 2×2 matrix, or a 2×1 column vector. If the neighborhood is a row vector, it must be at least 8 columns wide and contain no zeros.

Default: `ones(3,3)`

`LineBufferSize`

Size of the line memory buffer, specified as a scalar integer.

Choose a power of two that accommodates the number of active pixels in a horizontal line. If you specify a value that is not a power of two, the object uses the next largest power of two. The object allocates (*neighborhood lines* − 1)-by-`LineBufferSize` memory locations to store the pixels.

Default: `2048`

Methods

clone	Create object having the same property values
isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics

step	Report opened pixel value based on neighborhood

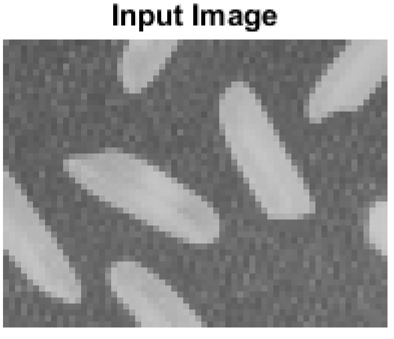
Examples

Grayscale Morphological Opening

Perform morphological opening on a grayscale thumbnail image.

Load a source image from a file. Select a portion of the image matching the desired test size.

```
frmOrig = imread('rice.png');
frmActivePixels = 64;
frmActiveLines = 48;
frmInput = frmOrig(1:frmActiveLines,1:frmActivePixels);
imshow(frmInput,'InitialMagnification',300)
title 'Input Image'
```



Create a serializer object and define the inactive pixel regions. Make the number of inactive pixels following each active line at least double the horizontal size of the neighborhood. Make the number of lines following each frame at least double the vertical size of the neighborhood.

```
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines,...
      'TotalPixelsPerLine',frmActivePixels+20,...
      'TotalVideoLines',frmActiveLines+20,...
      'StartingActiveLine',3,...
      'FrontPorch',10);
```

Create a filter object.

```
mopen = visionhdl.GrayscaleOpening( ...
          'Neighborhood',ones(2,7));
```

Serialize the test image by calling `step` on the serializer object. `pixIn` is a vector of intensity values. `ctrlIn` is a vector of control signal structures.

```
[pixIn,ctrlIn] = step(frm2pix,frmInput);
```

Prepare to process pixels by preallocating output vectors.

```
[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
pixOut = uint8(zeros(numPixelsPerFrame,1));
ctrlOut = repmat(pixelcontrolstruct,numPixelsPerFrame,1);
```

For each pixel in the padded frame, compute the morphed value. Monitor the control signals to determine the latency of the object. The latency of a configuration depends on the number of active pixels in a line and the size of the neighborhood.

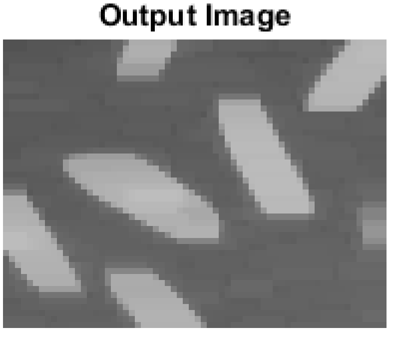
```
foundValIn = false;
foundValOut = false;
for p = 1:numPixelsPerFrame
    if (ctrlIn(p).valid && foundValIn==0)
        foundValIn = p;
    end
    [pixOut(p),ctrlOut(p)] = step(mopen,pixIn(p),ctrlIn(p));
    if (ctrlOut(p).valid && foundValOut==0)
        foundValOut = p;
    end
end
sprintf('object latency is %d cycles',foundValOut-foundValIn)
```

```
ans =
```

```
object latency is 222 cycles
```

Create a deserializer object with a format matching that of the serializer. Convert the pixel stream to an image frame by calling `step` on the deserializer object. Display the resulting image.

```
pix2frm = visionhdl.PixelsToFrame(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines);
[frmOutput,frmValid] = step(pix2frm,pixOut,ctrlOut);
if frmValid
    figure
    imshow(frmOutput, 'InitialMagnification',300)
    title 'Output Image'
end
```



Algorithm

This object implements the algorithms described on the `Grayscale Opening` block reference page.

See Also

visionhdl.GrayscaleDilation | visionhdl.GrayscaleClosing | `Grayscale Opening` | vision.MorphologicalOpen | `imopen` | visionhdl.GrayscaleErosion | visionhdl.FrameToPixels

Introduced in R2016a

clone

System object: visionhdl.GrayscaleOpening
Package: visionhdl

Create object having the same property values

Syntax

`newO = clone(O)`

Description

`newO = clone(O)` creates another instance of the `GrayscaleOpening` System object, `O`, that has the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

O

`visionhdl.GrayscaleOpening` System object

Output Arguments

newO

New `GrayscaleOpening` System object that has the same property values as the original System object.

Introduced in R2016a

isLocked

System object: visionhdl.GrayscaleOpening
Package: visionhdl

Locked status (logical)

Syntax

`TF = isLocked(O)`

Description

`TF = isLocked(O)` returns the locked status, `TF`, of the `GrayscaleOpening` System object, `O`.

Introduced in R2016a

release

System object: visionhdl.GrayscaleOpening
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

`release(O)`

Description

`release(O)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

O

`visionhdl.GrayscaleOpening` System object

Introduced in R2016a

step

System object: visionhdl.GrayscaleOpening
Package: visionhdl

Report opened pixel value based on neighborhood

Syntax

`[pixelOut,ctrlOut] = step(O,pixelIn,ctrlIn)`

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[pixelOut,ctrlOut] = step(O,pixelIn,ctrlIn)` returns the next pixel value, `pixelOut`, resulting from morphological opening on the neighborhood around each input pixel intensity value, `pixelIn`.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

O

visionhdl.GrayscaleOpening System object

pixelIn

Single pixel, specified as a scalar value.

Supported data types:

- uint8, uint16,uint32
- fixdt(0,N,M)
- double and single data types are supported for simulation but not for HDL code generation.

ctrlIn

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five logical signals. See “Pixel Control Structure”.

Output Arguments

pixelOut

Single pixel transformed by a morphological operation, returned as a scalar value.

The data type is the same as the data type of pixelIn.

ctrlOut

Control signals indicating the validity of the output pixel and the location of the pixel within the frame, returned as a structure containing five logical signals. See “Pixel Control Structure”.

Introduced in R2016a

visionhdl.PixelsToFrame System object

Package: visionhdl

Convert pixel stream to frame-based video

Description

`visionhdl.visionhdl.PixelsToFrame` converts a color or grayscale pixel stream and control structures into frame-based video. The control structure indicates the validity of each pixel and its location in the frame. The pixel stream format can include padding pixels around the active frame. You can configure the frame and padding dimensions by selecting a common video format or specifying custom dimensions. See "Streaming Pixel Interface" for details of the pixel stream format.

Use this object to convert the output of a function targeted for HDL code generation back to frames. This object does not support HDL code generation.

If your design converts frames to a pixel stream and later converts the stream back to frames, specify the same video format for the `FrameToPixels` object and the `PixelsToFrame` object.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`P2F = visionhdl.PixelsToFrame` returns a System object, `P2F`, that converts a 1080p pixel stream, with standard padding, to a grayscale 1080×1920 frame.

`P2F = visionhdl.PixelsToFrame(Name,Value)` returns a System object, `P2F`, with additional options specified by one or more `Name,Value` pair arguments. `Name` is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as Name1,Value1,...,NameN,ValueN. Properties not specified retain their default values.

Properties

NumComponents

Components of each pixel, specified as 1, 3, or 4. Set to 1 for grayscale video. Set to 3 for color video, for example, {R,G,B} or {Y,Cb,Cr}. Set to 4 to use color with an alpha channel for transparency. The step method expects a matrix of *P*-by-NumComponents values, where *P* is the total number of pixels. The default is 1.

VideoFormat

Dimensions of the active region of a video frame. To select a predefined format, specify the VideoFormat property as one of the options in the first column of the table. For a custom format, set VideoFormat to 'Custom', and specify the dimensional properties as integers.

Video Format	Active Pixels Per Line	Active Video Lines
240p	320	240
480p	640	480
480pH	720	480
576p	720	576
720p	1280	720
768p	1024	768
1024p	1280	1024
1080p (default)	1920	1080
1200p	1600	1200
2KCinema	2048	1080
4KUHDTV	3840	2160
8KUHDTV	7680	4320
Custom	User-defined	User-defined

Methods

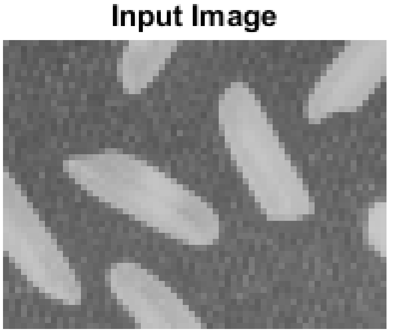
clone	Create object having the same property values
isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics
step	Convert pixel stream to image frame

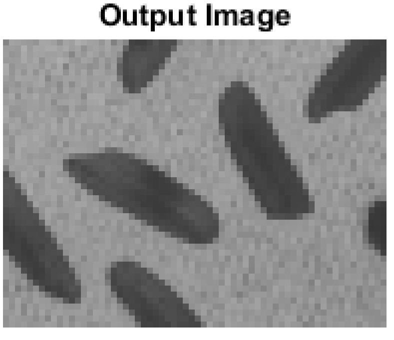
Examples

Convert Between Full-Frame and Pixel-Streaming Data

```
% This example converts a custom-size grayscale image to a pixel stream. It
% uses the visionhdl.LookupTable object to obtain the negative image. Then
% it converts the pixel-stream back to a full-frame image.

% Set the dimensions of the test image
frmActivePixels = 64;
frmActiveLines = 48;

% Load the source image
frmOrig = imread('rice.png');

% Select a portion of the image matching the desired test size
frmInput = frmOrig(1:frmActiveLines,1:frmActivePixels);
figure
imshow(frmInput,'InitialMagnification',300)
title 'Input Image'

% Create serializer and specify size of inactive pixel regions
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines,...
      'TotalPixelsPerLine',frmActivePixels+10,...
      'TotalVideoLines',frmActiveLines+10,...
      'StartingActiveLine',6,...
      'FrontPorch',5);
```

```
% Create LUT to generate the negative of the input image
tabledata = linspace(255,0,256);
inverter = visionhdl.LookupTable(tabledata);

% Serialize the test image
% pixIn is a vector of intensity values
% ctrlIn is a vector of control signal structures
[pixIn,ctrlIn] = step(frm2pix,frmInput);

% Prepare to process pixels
[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
pixOut = zeros(numPixelsPerFrame,1,'uint8');
ctrlOut = repmat(pixelcontrolstruct,numPixelsPerFrame,1);

% For each pixel in the stream, look up the negative of the pixel value
for p = 1:numPixelsPerFrame
    [pixOut(p),ctrlOut(p)] = step(inverter,pixIn(p),ctrlIn(p));
end

% Create deserializer with format matching that of the serializer
pix2frm = visionhdl.PixelsToFrame(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines);

% Convert the pixel stream to an image frame
[frmOutput,frmValid] = step(pix2frm,pixOut,ctrlOut);
if frmValid
    figure
    imshow(frmOutput,'InitialMagnification',300)
    title 'Output Image'
end
```





- “Pixel-Streaming Design in MATLAB”

See Also

visionhdl.FrameToPixels | `Pixels To Frame`

More About

- “Streaming Pixel Interface”

Introduced in R2015a

clone

System object: visionhdl.PixelsToFrame
Package: visionhdl

Create object having the same property values

Syntax

```
newP2F = clone(P2F)
```

Description

`newP2F = clone(P2F)` creates another instance of the `PixelsToFrame` System object, `P2F`, that has the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

`P2F`

`visionhdl.PixelsToFrame` System object

Output Arguments

`newP2F`

New `PixelsToFrame` System object that has the same property values as the original object.

Introduced in R2015a

isLocked

System object: visionhdl.PixelsToFrame
Package: visionhdl

Locked status (logical)

Syntax

```
TF = isLocked(P2F)
```

Description

`TF = isLocked(P2F)` returns the locked status, `TF`, of the `PixelsToFrame` System object, `P2F`.

Introduced in R2015a

release

System object: visionhdl.PixelsToFrame
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

`release(P2F)`

Description

`release(P2F)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

P2F

`visionhdl.PixelsToFrame` System object

Introduced in R2015a

step

System object: visionhdl.PixelsToFrame
Package: visionhdl

Convert pixel stream to image frame

Syntax

```
[frm,validOut] = step(P2F,pixels,ctrlIn)
```

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

```
[frm,validOut] = step(P2F,pixels,ctrlIn)
```

Converts a vector of pixel values representing a padded image, `pixels`, and an associated vector of control structures, `ctrlIn`, to an image matrix, `frm`. The control structure indicates the validity of each pixel and its location in the frame. The output image, `frm` is valid if `validOut` is `true`.

See "Streaming Pixel Interface" for details of the pixel stream format.

Note: The System object performs an initialization the first time you call the `step` method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the object issues an error. To change nontunable properties or inputs, first call the `release` method to unlock the object.

Input Arguments

P2F

visionhdl.PixelsToFrame System object

pixels

Pixel values, specified as a *P*-by-NumComponents matrix, where:

- *P* is the total number of pixels in the padded image, calculated as TotalPixelsPerLine × TotalVideoLines
- NumComponents is the number of components used to express a single pixel

Set the size of the padded image using the VideoFormat property. If the number of elements in pixels does not match that specified by VideoFormat, The object returns a warning.

Supported data types:

- uint or int
- fixdt()
- logical
- double or single

ctrlIn

Control structures associated with the input pixels, specified as a *P*-by-1 vector. *P* is the total number of pixels in the padded image, calculated as TotalPixelsPerLine × TotalVideoLines. Each structure contains five control signals indicating the validity of the pixel and its location in the frame. See “Pixel Control Structure”. If the dimensions indicated by ctrlIn do not match that specified by VideoFormat, the object returns a warning.

Output Arguments

frm

Image frame, returned as an ActiveVideoLines-by-ActivePixelsPerLine-by-NumComponents matrix, where:

- `ActiveVideoLines` is the height of the active image
- `ActivePixelsPerLine` is the width of the active image
- `NumComponents` is the number of components used to express a single pixel

Set the size of the active image using the `VideoFormat` property. The data type of the pixel values is the same as `pixels`.

validOut

Frame status, returned as a logical value. When `validOut` is `true`, the frame is reassembled and ready for use.

Introduced in R2015a

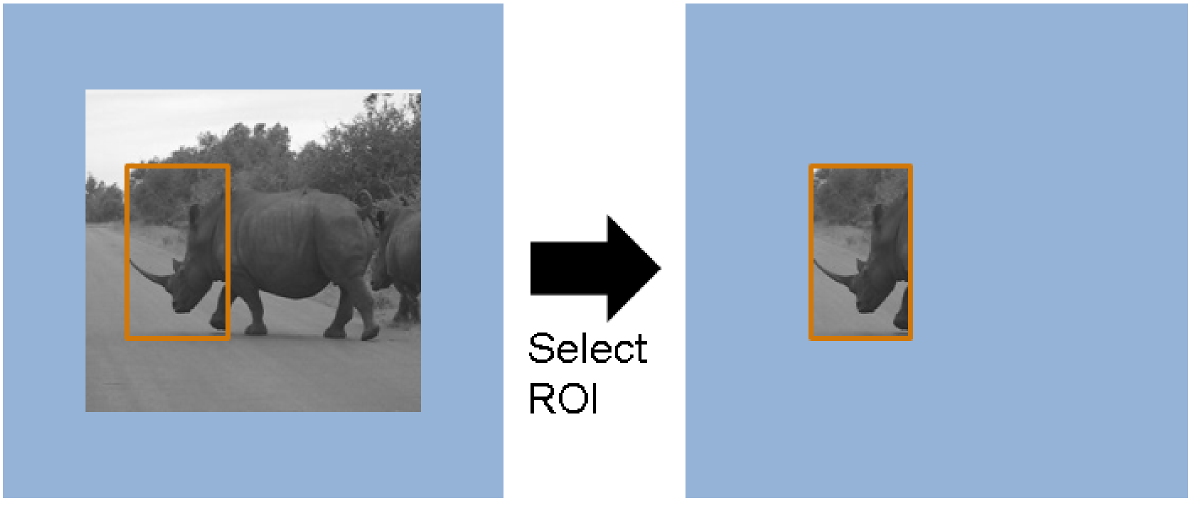
visionhdl.ROISelector System object

Package: visionhdl

Select region of interest (ROI) from pixel stream

Description

The `visionhdl.ROISelector` System object selects a portion of the active frame from a video stream. The total size of the frame remains the same. The control signals indicate a new active region of the frame. The diagram shows the inactive pixel regions in blue and the requested output region outlined in orange.



You can specify a fixed size and location for the new frame, or select the frame location in real time via an input argument. You can select more than one region. Define each region by the upper-left corner coordinates and the dimensions. The object returns one set of pixels and control signals for each region you specify. The object sets the inactive pixels in the output frame to zero. Regions are independent from each other, so they can overlap. If you specify a region that includes the edge of the active frame, the object returns only the active portion of the region. The diagram shows the output frames for three requested regions. The second output region (treetops) does not include the inactive region above the frame.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The `step` method accepts and returns a scalar pixel value. `step` also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`ROI = visionhdl.ROISelector` returns a System object, `ROI`, that selects a default region of the active frame from an input stream.

`ROI = visionhdl.ROISelector(Name,Value)` returns a System object, `ROI`, with additional options specified by one or more `Name,Value` pair arguments. `Name` is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Properties not specified retain their default values.

Properties

`RegionsSource`

Location of the output region definitions.

`'Property'` — Specify the regions in the `Regions` property.

`'Input port'` — Specify the regions using arguments to the `step` method. Each argument is a 1-by-4 vector specifying coordinates for a single region. The object captures the value of the `region` input ports when it receives `vStart` set to `true` in the input control structure.

Default: `'Property'`

`Regions`

Rectangular regions of interest to select from the input frame, specified as a *N*-by-4 matrix.

N is the number of regions. You can select up to 16 regions. The four elements that define each region are the top-left starting coordinates and the dimensions, `[hPos vPos hSize vSize]`. The coordinates count from the upper-left corner of the active frame, defined as [1,1]. `hSize` must be greater than 1. The regions are independent of each other, so they can overlap. This property applies when you set `RegionsSource` to `'Property'`.

Default: `[100 100 50 50]`

`NumberofRegions`

Number of `region` arguments to the `step` method, specified as a positive integer.

You can select up to 16 regions. This property applies when you set `RegionsSource` to `'Input port'`.

Default: `1`

Methods

clone	Create object having the same property values

isLocked	Locked status (logical)
release	Allow changes to property values and input characteristics
step	Return next pixel in reselected frame

Examples

Select Region of Interest

Select a fixed region of interest (ROI) from an input frame.

Load a source image from a file.

```
frmOrig = imread('coins.png');
[frmActiveLines,frmActivePixels] = size(frmOrig);
imshow(frmOrig)
title 'Input Image'
```

Create a serializer object and define inactive pixel regions.

```
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',frmActivePixels,...
      'ActiveVideoLines',frmActiveLines,...
      'TotalPixelsPerLine',frmActivePixels+20,...
      'TotalVideoLines',frmActiveLines+20,...
      'StartingActiveLine',3,...
      'FrontPorch',10);
```

Create an object to select a region of interest. Define a rectangular region by the coordinates of the top-left corner and the dimensions.

```
hPos = 80;
vPos = 60;
hSize = 65;
vSize = 50;
roicoin = visionhdl.ROISelector('Regions',[hPos vPos hSize vSize])
```

```
roicoin =

  visionhdl.ROISelector with properties:

    RegionsSource: 'Property'
          Regions: [80 60 65 50]
```

Serialize the test image by calling `step` on the serializer object. `pixIn` is a vector of intensity values. `ctrlIn` is a vector of control signal structures.

```
[pixIn,ctrlIn] = step(frm2pix,frmOrig);
```

Prepare to process pixels by preallocating output vectors. The output frame is the same size as the input frame, but the control signals indicate a different active region.

```
[~,~,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix);
pixOut = uint8(zeros(numPixelsPerFrame,1));
ctrlOut = repmat(pixelcontrolstruct,numPixelsPerFrame,1);
```

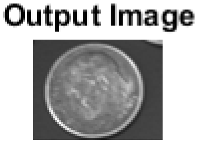
For each pixel in the padded frame, apply the region mask.

```
for p = 1:numPixelsPerFrame
```

```
    [pixOut(p),ctrlOut(p)] = step(roicoin,pixIn(p),ctrlIn(p));
end
```

Create a deserializer object with format matching the new region. Convert the pixel stream to an image frame by calling `step` on the deserializer object. Display the resulting image.

```
pix2frm = visionhdl.PixelsToFrame(...
      'NumComponents',1,...
      'VideoFormat','custom',...
      'ActivePixelsPerLine',hSize,...
      'ActiveVideoLines',vSize);
[frmOutput,frmValid] = step(pix2frm,pixOut,ctrlOut);
if frmValid
    figure
    imshow(frmOutput)
    title 'Output Image'
end
```



Algorithm

The generated HDL code for the `visionhdl.ROISelector` System object uses two 32-bit counters. It does not use additional counters for additional regions.

Latency

The object has a latency of three cycles. The object returns the output pixel and associated control signals on the third call to the `step` method after the pixel value was applied.

See Also

`ROI Selector` | visionhdl.FrameToPixels

Introduced in R2016a

clone

System object: visionhdl.ROISelector
Package: visionhdl

Create object having the same property values

Syntax

`newROI = clone(ROI)`

Description

`newROI = clone(ROI)` creates another instance of the `ROISelector` System object, `ROI`, that has the same property values. The new object is unlocked and contains uninitialized states.

Input Arguments

`ROI`

`visionhdl.ROISelector` System object

Output Arguments

`newROI`

New `ROISelector` System object that has the same property values as the original System object.

Introduced in R2016a

isLocked

System object: visionhdl.ROISelector
Package: visionhdl

Locked status (logical)

Syntax

```
TF = isLocked(ROI)
```

Description

`TF = isLocked(ROI)` returns the locked status, `TF`, of the `ROISelector` System object, `ROI`.

Introduced in R2016a

release

System object: visionhdl.ROISelector
Package: visionhdl

Allow changes to property values and input characteristics

Syntax

`release(ROI)`

Description

`release(ROI)` releases system resources (such as memory, file handles, or hardware connections), allowing you to change System object properties and input characteristics.

Input Arguments

`ROI`

`visionhdl.ROISelector` System object

Introduced in R2016a

step

System object: visionhdl.ROISelector
Package: visionhdl

Return next pixel in reselected frame

Syntax

```
[pixel1,ctrl1] = step(ROI,pixelIn,ctrlIn)
[pixel1,ctrl1,...,pixelN,ctrlN] = step(ROI,pixelIn,ctrlIn)
[pixel1,ctrl1,...,pixelN,ctrlN] =
step(ROI,pixelIn,ctrlIn,region1,...,regionN)
```

Description

Note: Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[pixel1,ctrl1] = step(ROI,pixelIn,ctrlIn)` returns the next pixel value, `pixel1`, and control signals, `ctrl1`, resulting from masking the active image frame into a single new region. Define the region by setting the `Regions` property to a 1-by-4 vector, `[hPos vPos hSize vSize]`.

`[pixel1,ctrl1,...,pixelN,ctrlN] = step(ROI,pixelIn,ctrlIn)` returns the next pixel values, `pixel1,...,pixelN`, and control signals, `ctrl1,...,ctrlN`, of each stream resulting from masking the active image frame into 1 to *N* new active regions, as directed by the `Regions` property. Set the `Regions` property to a *N*-by-4 matrix of region coordinates.

`[pixel1,ctrl1,...,pixelN,ctrlN] = step(ROI,pixelIn,ctrlIn,region1,...,regionN)` returns the next pixel values of each stream, `pixel1,...,pixelN`, resulting from masking the active image frame into 1 to *N* new regions, as directed by the `region1,...,regionN` arguments. Each region input is a 1-by-4 vector of region coordinates. Use this syntax when you set the RegionsSource property to 'Input Port', and the NumberOfRegions property to *N*.

This object uses a streaming pixel interface with a structure for synchronization control signals. This interface enables the object to operate independently of image size and format, and to connect easily with other Vision HDL Toolbox objects. The step method accepts and returns a scalar pixel value. step also accepts and returns control signals as a structure containing five signals. These signals indicate the validity of each pixel and the location of each pixel in the frame.

Note: The object performs an initialization the first time you call the step method. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, first call the release method to unlock the object.

Input Arguments

ROI

visionhdl.ROISelector System object

pixelIn

Input pixel, specified as a scalar integer value.

- Any numeric data type.

double and single data types are supported for simulation but not for HDL code generation.

ctrlIn

Control signals indicating the validity of the input pixel and the location of the pixel within the frame, specified as a structure containing five logical signals. See “Pixel Control Structure”.

region1,...,regionN

Regions of interest, specified as a 1-by-4 vector of coordinates.

The four elements that define each region are `[hPos vPos hSize vSize]`. Use this argument when `RegionsSource` is set to `'Input port'`. You can specify *N* regions, where *N* is the `NumberOfRegions` property value.

Output Arguments

`pixel1,...,pixelN`

Output pixels, specified as 1 to *N* scalar integers.

If you set `RegionsSource` to `'Input port'`, *N* is the value in `NumberOfRegions`. If you set `RegionsSource` to `'Property'`, *N* is the number of columns in the `Regions` property.

`ctrl1,...,ctrlN`

Control signals indicating the validity of each output pixel and the location of each pixel within the frame, returned as 1 to *N* structures of five `logical` signals. See "Pixel Control Structure".

If you set `RegionsSource` to `'Input port'`, *N* is the value in `NumberOfRegions`. If you set `RegionsSource` to `'Property'`, *N* is the number of columns in the `Regions` property.

Introduced in R2016a

Functions — Alphabetical List

getparamfromfrm2pix

Get frame format parameters

Syntax

```
[activePixelsPerLine,activeLines,numPixelsPerFrame] =
getparamfromfrm2pix(frm2pix)
```

Description

`[activePixelsPerLine,activeLines,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix)` returns video format parameters from a `visionhdl.FrameToPixels` System object.

Examples

Configure Pixel Stream Format

When you choose a standard video format for `visionhdl.FrameToPixels`, the object computes the frame dimensions. To access these values, call the `getparamfromfrm2pix` function.

```
frm2pix = visionhdl.FrameToPixels(...
      'NumComponents',1,...
      'VideoFormat','240p');
[activePixels,activeLines,numPixelsPerFrame] = getparamfromfrm2pix(frm2pix)
```

```
activePixels =

   320


activeLines =

   240
```

```
numPixelsPerFrame =

      130248
```

Note that `numPixelsPerFrame` includes both active and inactive regions of the frame.

Input Arguments

`frm2pix` — Video serializer

`visionhdl.FrameToPixels` System object

The `visionhdl.FrameToPixels` object converts framed video to a stream of pixel values and control signals. It contains useful parameters regarding the video format.

Output Arguments

`activePixelsPerLine` — Number of pixels in a horizontal line of the active video frame

positive integer

Number of pixels in a horizontal line of the active video frame, returned as a positive integer.

For custom video formats, this value corresponds to the `ActivePixelsPerLine` property of the `frm2pix` object.

`activeLines` — Number of horizontal lines in the active video frame

positive integer

Number of horizontal lines in the active video frame, returned as a positive integer.

For custom video formats, this value corresponds to the `ActiveVideoLines` property of the `frm2pix` object.

`numPixelsPerFrame` — Number of active and inactive pixels in the video frame

positive integer

Number of active and inactive pixels in the video frame, returned as a positive integer.

For custom video formats, this value corresponds to the product of the `TotalVideoLines` and `TotalPixelsPerLine` properties of the `frm2pix` object.

More About

- “Streaming Pixel Interface”

See Also

`Frame To Pixels` | `Pixels To Frame`

Introduced in R2015a

pixelcontrolbus

Create pixel-streaming control bus instance

Syntax

```
pixelcontrolbus
```

Description

`pixelcontrolbus` declares a bus instance in the workspace. This instance is required for HDL code generation. Call this function before you generate HDL code from Vision HDL Toolbox blocks.

Examples

Declare Bus in Base Workspace

In the `InitFcn` callback function of your Simulink model, include this line to declare a bus instance in the base workspace. If you create your model with the Vision HDL Toolbox model template, this is done for you.

```
evalin('base','pixelcontrolbus')
```

If you do not declare an instance of `pixelcontrolbus` in the base workspace, you might encounter this error when you generate HDL code in Simulink.

```
Cannot resolve variable 'pixelcontrol'
```

More About

- “Configure the Simulink Environment for HDL Video Processing”
- “Streaming Pixel Interface”

See Also

“Pixel Control Bus” | `Frame To Pixels` | `Pixels To Frame`

Introduced in R2015a

pixelcontrolsignals

Extract signals from pixel-streaming control signal structure

Syntax

```
hStart,hEnd,vStart,vEnd,valid] = pixelcontrolsignals(ctrl)
```

Description

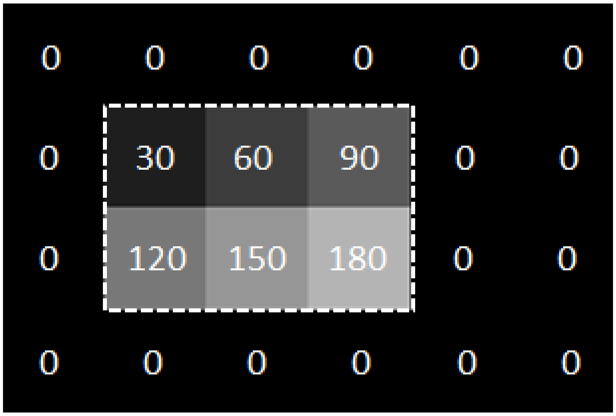
`hStart,hEnd,vStart,vEnd,valid] = pixelcontrolsignals(ctrl)` extracts five scalar `logical` control signals from a structure.

Examples

Create and Decompose pixelcontrol structures

If you integrate Vision HDL Toolbox designs with algorithms that use a different interface, you may need to create the structure manually, or manipulate the control signals outside of the structure.

Create a `pixelcontrol` structure by passing five control signal values to the `pixelcontrolstruct` function. The function arguments must be scalar values. These control signals may come from a camera or other video input source. The control signal vectors in this example describe a simple 2-by-3 pixel test image, surrounded by padding pixels.




```
hStart = [0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0];
vStart = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];
hEnd   = [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0];
vEnd   = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0];
valid  = [0 0 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0];
pixel = uint8([0 0 0 0 0 0 0 30 60 90 0 0 0 120 150 180 0 0 0 0 0 0 0 0]);
[~,numPix] = size(pixel);
ctrlIn = repmat(pixelcontrolstruct,numPix,1);
for i=1:numPix
    ctrlIn(i) = pixelcontrolstruct(hStart(i),vStart(i),...
                                   hEnd(i),vEnd(i),valid(i));
end
```

Each element of `ctrlIn` is a structure containing the five control signals.

```
ctrlIn(8)
```

```
ans = 

  struct with fields:

    hStart: 1
      hEnd: 1
    vStart: 0
      vEnd: 0
     valid: 1
```

You can then pass this structure to the `step` method of a Vision HDL Toolbox System object. This example uses the LookupTable object to invert each pixel.

```
tabledata = uint8(linspace(255,0,256));
inverter = visionhdl.LookupTable(tabledata);
pixelOut = zeros(numPix,1,'uint8');
ctrlOut = repmat(pixelcontrolstruct,numPix,1);
for i=1:numPix
    [pixelOut(i),ctrlOut(i)] = step(inverter,pixel(i),ctrlIn(i));
end
```

If you need to use the control signals directly in downstream algorithms, you can flatten each structure into five `logical` control signal values by calling the `pixelcontrolsignals` function.

```
[hStartOut,vStartOut,hEndOut,vEndOut,validOut] = deal(false(numPix,1));
for i=1:numPix
    [hStartOut(i),vStartOut(i),hEndOut(i),vEndOut(i),validOut(i)] = ...
        pixelcontrolsignals(ctrlOut(i));
end
```

Each output control signal is a vector of logical values that correspond with the `pixelOut` vector.

```
validOut'
```

```
ans =

  1×24 logical array

  Columns 1 through 19

   0   0   0   0   0   0   0   0   0   1   1   1   0   0   0   1   1   1   0

  Columns 20 through 24

   0   0   0   0   0
```

Input Arguments

`ctrl` — Pixel control signals

structure containing five `logical` values

Pixel control signals, specified as a structure containing five `logical` values.

The pixel control structure is a specific format used by Vision HDL Toolbox objects. See “Pixel Control Structure”.

Output Arguments

`hStart` — Control signal indicating the first pixel in a horizontal line
`logical`

Control signal indicating the first pixel in a horizontal line, specified as a `logical` scalar.

`hEnd` — Control signal indicating the last pixel in a horizontal line
`logical`

Control signal indicating the last pixel in a horizontal line, specified as a `logical` scalar.

`vStart` — Control signal indicating the first pixel in the first (top) line
`logical`

Control signal indicating the first pixel in the first (top) line, specified as a `logical` scalar.

`vEnd` — Control signal indicating the last pixel in the last (bottom) line
`logical`

Control signal indicating the last pixel in the last (bottom) line, specified as a `logical` scalar.

`valid` — Control signal indicating the validity of the pixel
`logical`

Control signal indicating the validity of the pixel, specified as a `logical` scalar.

More About

- “Streaming Pixel Interface”

See Also

visionhdl.FrameToPixels | visionhdl.PixelsToFrame | `pixelcontrolstruct`

Introduced in R2015a

pixelcontrolstruct

Create pixel-streaming control signal structure

Syntax

```
ctrl = pixelcontrolstruct(hStart,hEnd,vStart,vEnd,valid)
```

Description

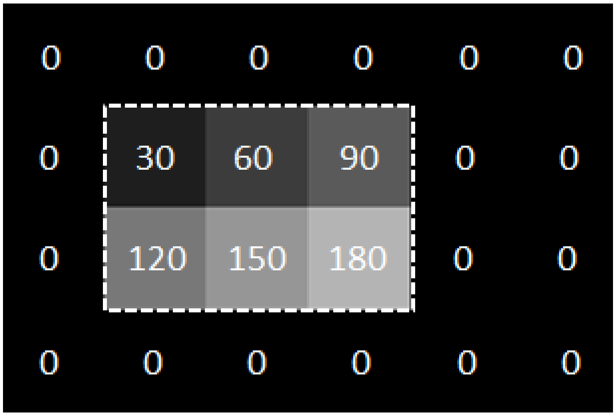
`ctrl = pixelcontrolstruct(hStart,hEnd,vStart,vEnd,valid)` creates a structure containing the five control signals used by Vision HDL Toolbox objects. The input arguments must be five scalars of `logical` type. See "Pixel Control Structure".

Examples

Create and Decompose pixelcontrol structures

If you integrate Vision HDL Toolbox designs with algorithms that use a different interface, you may need to create the structure manually, or manipulate the control signals outside of the structure.

Create a `pixelcontrol` structure by passing five control signal values to the `pixelcontrolstruct` function. The function arguments must be scalar values. These control signals may come from a camera or other video input source. The control signal vectors in this example describe a simple 2-by-3 pixel test image, surrounded by padding pixels.




```
hStart = [0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0];
vStart = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];
hEnd   = [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0];
vEnd   = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0];
valid  = [0 0 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0];
pixel = uint8([0 0 0 0 0 0 0 30 60 90 0 0 0 120 150 180 0 0 0 0 0 0 0 0]);
[~,numPix] = size(pixel);
ctrlIn = repmat(pixelcontrolstruct,numPix,1);
for i=1:numPix
    ctrlIn(i) = pixelcontrolstruct(hStart(i),vStart(i),...
                                   hEnd(i),vEnd(i),valid(i));
end
```

Each element of `ctrlIn` is a structure containing the five control signals.

```
ctrlIn(8)
```

```
ans = 

  struct with fields:

    hStart: 1
      hEnd: 1
    vStart: 0
      vEnd: 0
     valid: 1
```

You can then pass this structure to the `step` method of a Vision HDL Toolbox System object. This example uses the LookupTable object to invert each pixel.

```
tabledata = uint8(linspace(255,0,256));
inverter = visionhdl.LookupTable(tabledata);
pixelOut = zeros(numPix,1,'uint8');
ctrlOut = repmat(pixelcontrolstruct,numPix,1);
for i=1:numPix
    [pixelOut(i),ctrlOut(i)] = step(inverter,pixel(i),ctrlIn(i));
end
```

If you need to use the control signals directly in downstream algorithms, you can flatten each structure into five `logical` control signal values by calling the `pixelcontrolsignals` function.

```
[hStartOut,vStartOut,hEndOut,vEndOut,validOut] = deal(false(numPix,1));
for i=1:numPix
    [hStartOut(i),vStartOut(i),hEndOut(i),vEndOut(i),validOut(i)] = ...
        pixelcontrolsignals(ctrlOut(i));
end
```

Each output control signal is a vector of logical values that correspond with the `pixelOut` vector.

```
validOut'
```

```
ans =

  1×24 logical array

  Columns 1 through 19

   0   0   0   0   0   0   0   0   0   1   1   1   0   0   0   1   1   1   0

  Columns 20 through 24

   0   0   0   0   0
```

Input Arguments

`hStart` — Control signal indicating the first pixel in a horizontal line

`logical`

Control signal indicating the first pixel in a horizontal line, specified as a `logical` scalar.

`hEnd` — Control signal indicating the last pixel in a horizontal line
`logical`

Control signal indicating the last pixel in a horizontal line, specified as a `logical` scalar.

`vStart` — Control signal indicating the first pixel in the first (top) line
`logical`

Control signal indicating the first pixel in the first (top) line, specified as a `logical` scalar.

`vEnd` — Control signal indicating the last pixel in the last (bottom) line
`logical`

Control signal indicating the last pixel in the last (bottom) line, specified as a `logical` scalar.

`valid` — Control signal indicating the validity of the pixel
`logical`

Control signal indicating the validity of the pixel, specified as a `logical` scalar.

Output Arguments

`ctrl` — Pixel control signals
structure containing five `logical` values

Pixel control signals, specified as a structure containing five `logical` values.

The pixel control structure is a specific format used by Vision HDL Toolbox objects. See “Pixel Control Structure”.

More About

- “Streaming Pixel Interface”

See Also

visionhdl.FrameToPixels | visionhdl.PixelsToFrame | `pixelcontrolsignals`

Introduced in R2015a