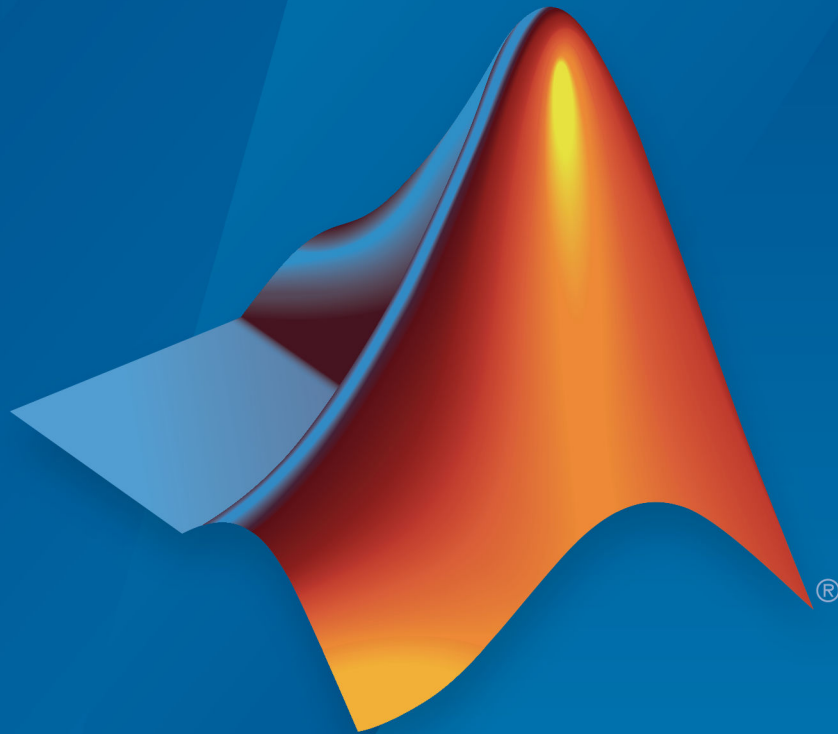


Vision HDL Toolbox™
Getting Started Guide



MATLAB®

R2017b



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™ Getting Started Guide

© COPYRIGHT 2015–2017 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

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)
March 2017	Online only	Revised for Version 1.4 (Release R2017a)
September 2017	Online only	Revised for Version 1.5 (Release R2017b)

Vision HDL Toolbox Getting Started

Vision HDL Toolbox Product Description	1-2
Key Features	1-2
Design Video Processing Algorithms for HDL in Simulink ..	1-3
Open Model Template	1-4
Import Data	1-5
Serialize Data	1-5
Design HDL-Compatible Model	1-6
Design Behavioral Model	1-8
Deserialize Filtered Pixel Stream	1-8
Display Results and Compare to Behavioral Model	1-8
Generate HDL Code	1-9
Design Video Processing Algorithms for HDL in MATLAB .	1-10
Import Data	1-11
Serialize Data	1-12
Design HDL-Compatible Model	1-13
Deserialize Filtered Pixel Stream	1-14
Display Results	1-15
Compare to Behavioral Model	1-15
HDL Code Generation	1-16
Configure the Simulink Environment for HDL Video	
Processing	1-17
About Simulink Model Templates	1-17
Create Model Using Vision HDL Toolbox Model Template ..	1-17
Vision HDL Toolbox Model Template	1-18

Vision HDL Toolbox Getting Started

- “Vision HDL Toolbox Product Description” on page 1-2
- “Design Video Processing Algorithms for HDL in Simulink” on page 1-3
- “Design Video Processing Algorithms for HDL in MATLAB” on page 1-10
- “Configure the Simulink Environment for HDL Video Processing” on page 1-17

Vision HDL Toolbox Product Description

Design image processing, video, and computer vision systems for FPGAs and ASICs

Vision HDL Toolbox provides pixel-streaming algorithms for the design and implementation of vision systems on FPGAs and ASICs. It provides a design framework that supports a diverse set of interface types, frame sizes, and frame rates, including high-definition (1080p) video. The image processing, video, and computer vision algorithms in the toolbox use an architecture appropriate for HDL implementations.

The toolbox algorithms are designed to generate readable, synthesizable code in VHDL and Verilog (with HDL Coder™). The generated HDL code can process 1080p60 in real time.

Toolbox capabilities are available as MATLAB® System objects and Simulink® blocks.

Key Features

- Video synchronization signal controls for handling nonideal timing and resolution variations
- Configurable frame rates and sizes, including 60FPS for high-definition (1080p) video
- Frame-to-pixel and pixel-to-frame conversions to integrate with frame-based processing capabilities in MATLAB and Simulink
- Image processing, video, and computer vision algorithms with a pixel-streaming architecture, including image enhancement, filtering, morphology, and statistics
- Implicit onchip data handling using line memory
- Support for HDL code generation and real-time verification

Design Video Processing Algorithms for HDL in Simulink

This tutorial shows how to design a hardware-targeted image filter using Vision HDL Toolbox blocks. It also uses Computer Vision System Toolbox™ blocks.

The key features of a model for hardware-targeted video processing in Simulink are:

- **Streaming pixel interface**

Blocks in Vision HDL Toolbox use a streaming pixel interface. Serial processing is efficient for hardware designs, because less memory is required to store pixel data for computation. The serial interface allows the block to operate independently of image size and format and makes the design more resilient to video timing errors. For further information, see “Streaming Pixel Interface”.

- **Subsystem targeted for HDL code generation**

Design a hardware-friendly pixel-streaming video processing model by selecting blocks from the Vision HDL Toolbox libraries. The part of the design targeted for HDL code generation must be in a separate subsystem.


- **Conversion to frame-based video**

For verification, you can display frame-based video or compare the result of your hardware-compatible design with the output of a Simulink behavioral model. Vision HDL Toolbox provides a block that allows you to deserialize the output of your design.

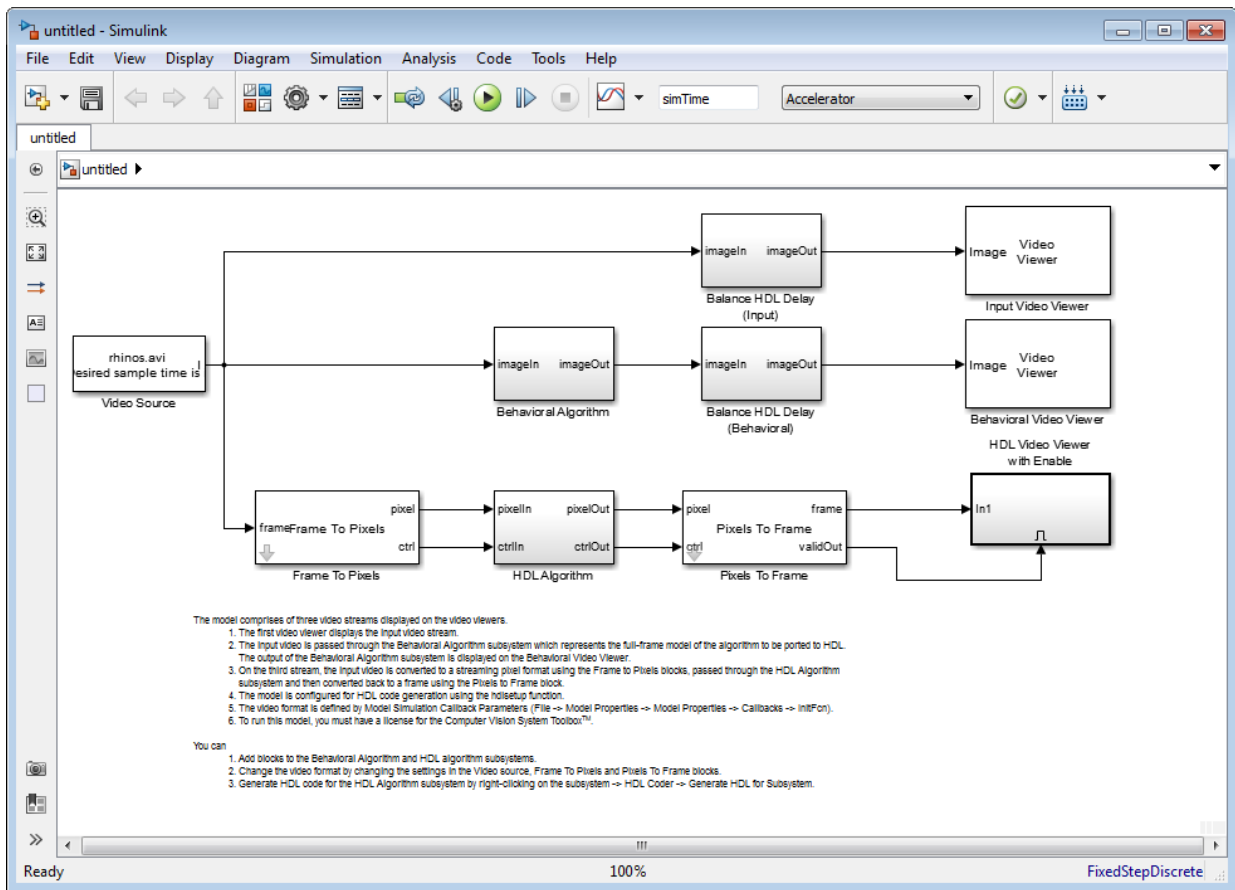
In this section...
“Open Model Template” on page 1-4
“Import Data” on page 1-5
“Serialize Data” on page 1-5
“Design HDL-Compatible Model” on page 1-6
“Design Behavioral Model” on page 1-8
“Deserialize Filtered Pixel Stream” on page 1-8
“Display Results and Compare to Behavioral Model” on page 1-8
“Generate HDL Code” on page 1-9

Open Model Template

This tutorial uses a Simulink model template to get started.

- 1 Click the Simulink button, , or type `simulink` at the MATLAB command prompt.
- 2 On the Simulink start page, find the Vision HDL Toolbox section, and click the **Basic Model** template.

The template creates a new model that you can customize. Save the model with a new name.



Import Data

The template includes a Video Source block that contains a 240p video sample. Each pixel is a scalar `uint8` value representing intensity. A best practice is to design and debug your design using a small frame size for quick debug cycles, before scaling up to larger image sizes. You can use this 240p source to debug a design targeted for 1080p video.

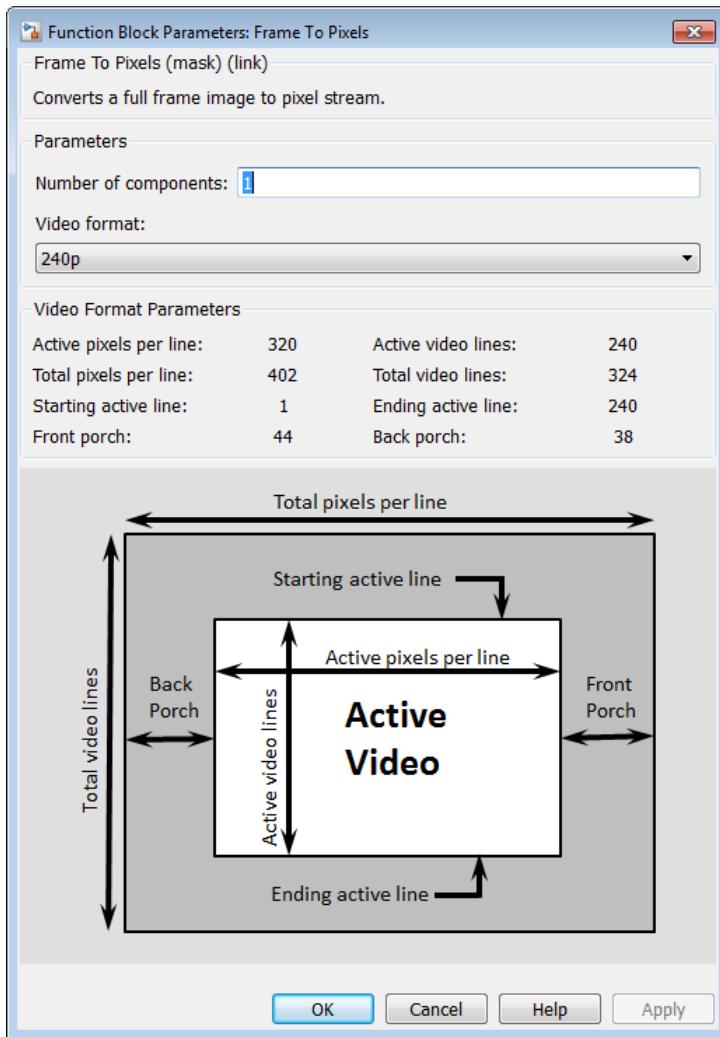
Serialize Data

The Frame To Pixels block converts framed video to a stream of pixels and control structures. This block provides input for a subsystem targeted for HDL code generation, but it does not itself support HDL code generation.

The template includes an instance of this block. To simulate with a standard video format, choose a predefined video padding format to match your input source. To simulate with a custom-size image, choose the dimensions of the inactive regions that you want to surround the image with. This tutorial uses a standard video format.

Open the Frame To Pixels block dialog box to view the settings. The source video is in 240p grayscale format. A scalar integer represents the intensity value of each pixel. To match the input video, set **Number of components** to 1, and the **Video format** to 240p.

Note The sample time of the 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*. In the `InitFcn` callback, the template creates a workspace variable, `totalPixels`, for the sample time of a 240p frame.

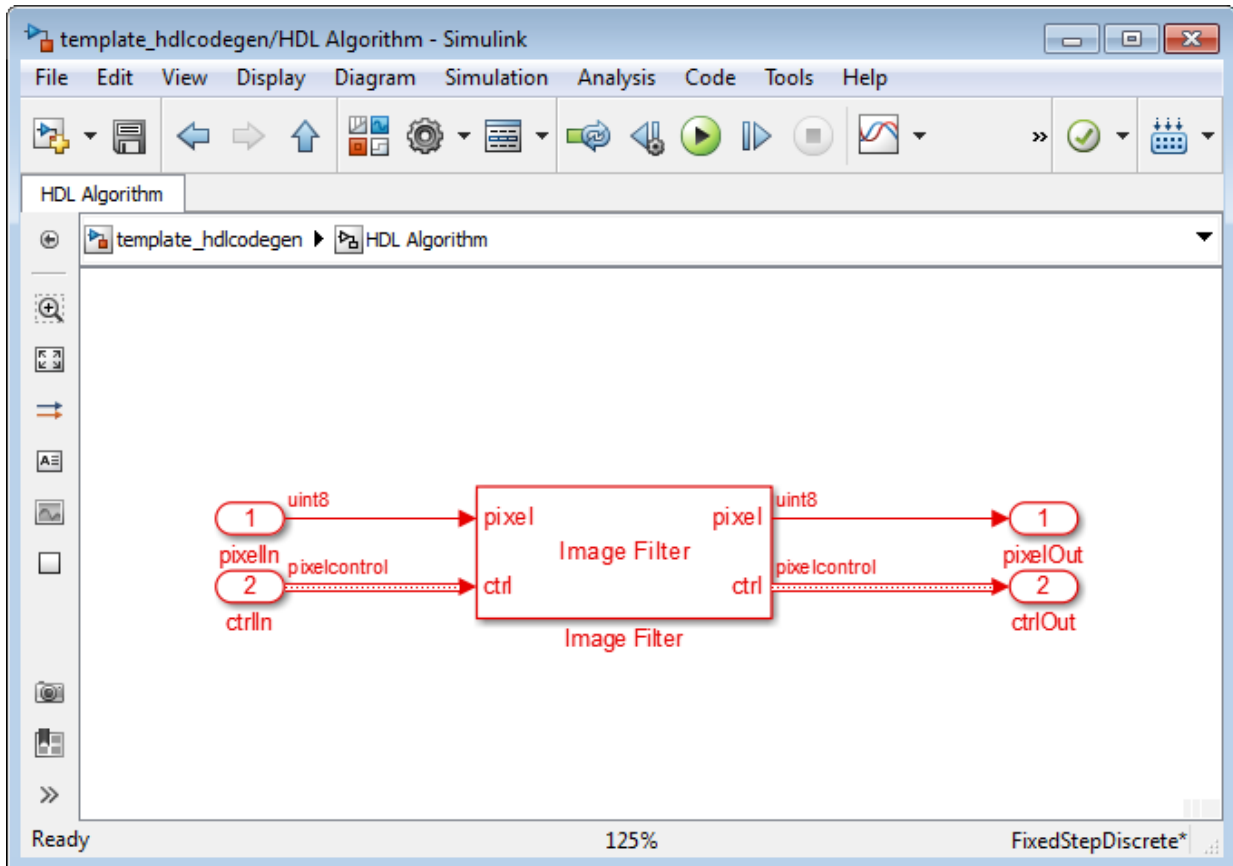


Design HDL-Compatible Model

Design a subsystem targeted for HDL code generation, by modifying the HDL Algorithm subsystem. The subsystem input and output ports use the streaming pixel format described in the previous section. Open the HDL Algorithm subsystem to edit it.

In the **Simulink Library Browser**, click **Vision HDL Toolbox**. You can also open this library by typing `visionhdl1lib` at the MATLAB command prompt.

Select an image processing block. This example uses the Image Filter block from the Filtering sublibrary. You can also access this library by typing `visionhdlfilter` at the MATLAB command prompt. Add the Image Filter block to the HDL Algorithm subsystem and connect the ports.



Open Image Filter block and make the following changes:

- Set **Filter coefficients** to `ones(4,4)/16` to implement a 4×4 blur operation.
- Set **Padding method** to `Symmetric`.

- Set **Line buffer size** to a power of 2 that accommodates the active line size of the largest required frame format. This parameter does not affect simulation speed, so it does not need to be reduced when simulating with a small test image. The default, 2048, accommodates 1080p video format.
- On the **Data Types** tab, under **Data Type**, set **Coefficients** to `fixdt(0,1,4)`.

Design Behavioral Model

You can visually or mathematically compare your HDL-targeted design with a behavioral model to verify the hardware design and monitor quantization error. The template includes a Behavioral Model subsystem with frame-based input and output ports for this purpose. Double-click on the Behavioral Model to edit it.

For this tutorial, add the 2-D FIR Filter block from Computer Vision System Toolbox. This block filters the entire frame at once.

Open the 2-D FIR Filter block and make the following changes to match the configuration of the Image Filter block from Vision HDL Toolbox:

- Set **Coefficients** to `ones(4,4)/16` to implement a 4×4 blur operation.
- Set **Padding options** to `Symmetric`.
- On the **Data Types** tab, under **Data Type**, set **Coefficients** to `fixdt(0,2,4)`.

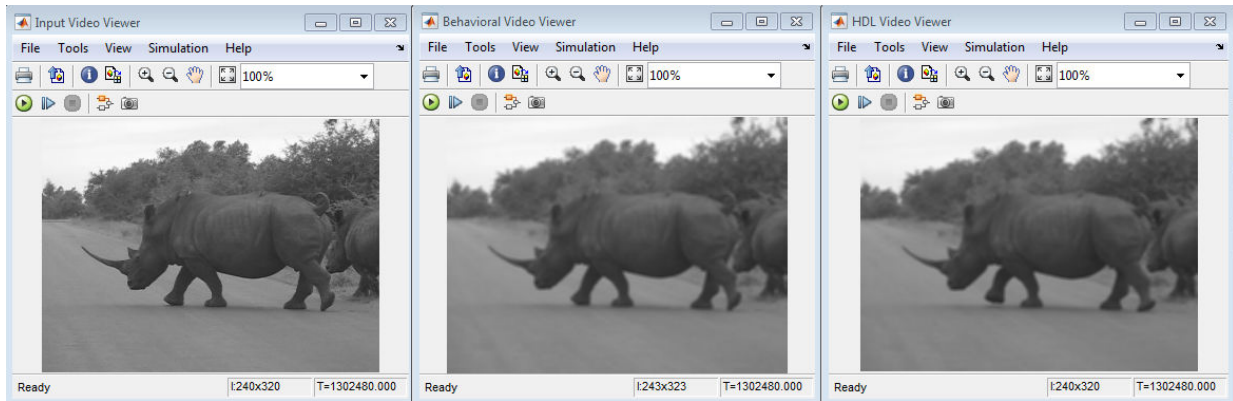
Deserialize Filtered Pixel Stream

Use the Pixels To Frame block included in the template to deserialize the data for display.

Open the Pixels To Frame block. Set the image dimension properties to match the input video and the settings you specified in the Frame To Pixels block. For this tutorial, the **Number of components** is set to 1 and the **Video format** is set to 240p. The block converts the stream of output pixels and control signals back to a matrix representing a frame.

Display Results and Compare to Behavioral Model

Use the Video Viewer blocks included in the template to compare the output frames visually. The `validOut` signal of the Pixels To Frame block is connected to the `Enable` port of the viewer. Run the model to display the results.



Generate HDL Code

Once your design is working in simulation, you can use HDL Coder to generate HDL code for the HDL Algorithm subsystem. See “Generate HDL Code From Simulink”.

See Also

Related Examples

- “Gamma Correction”

More About

- “Configure the Simulink Environment for HDL Video Processing” on page 1-17

Design Video Processing Algorithms for HDL in MATLAB

This tutorial shows how to design a hardware-targeted image filter using Vision HDL Toolbox objects.

The key features of a model for hardware-targeted video processing in MATLAB are:

- **Streaming pixel interface**

System objects in Vision HDL Toolbox use a streaming pixel interface. Serial processing is efficient for hardware designs, because less memory is required to store pixel data. The serial interface enables the object to operate independently of image size and format and makes the design more resilient to video timing errors. For further information, see “Streaming Pixel Interface”.

- **Function targeted for HDL code generation**

Once the data is converted to a pixel stream, you can design a hardware model by selecting System objects from the Vision HDL Toolbox libraries. The part of the design targeted for HDL code generation must be in a separate function.

- **Conversion to frame-based video**

For verification, you can display frame-based video, or you can compare the result of your hardware-compatible design with the output of a MATLAB frame-based behavioral model. Vision HDL Toolbox provides a System object™ that enables you to deserialize the output of your design.

In this section...
“Import Data” on page 1-11
“Serialize Data” on page 1-12
“Design HDL-Compatible Model” on page 1-13
“Deserialize Filtered Pixel Stream” on page 1-14
“Display Results” on page 1-15
“Compare to Behavioral Model” on page 1-15
“HDL Code Generation” on page 1-16

Import Data

Read an image file into the workspace. This sample image contains 256×256 pixels. Each pixel is a single `uint8` value representing intensity. To reduce simulation speed while testing, select a thumbnail portion of the image.

```
origIm = imread('rice.png');  
origImSize = size(origIm)  
imActivePixels = 64;  
imActiveLines = 48;  
inputIm = origIm(1:imActiveLines,1:imActivePixels);  
figure  
imshow(inputIm, 'InitialMagnification',300)  
title 'Input Image'
```

```
origImSize =  
  
    256    256
```

Input Image

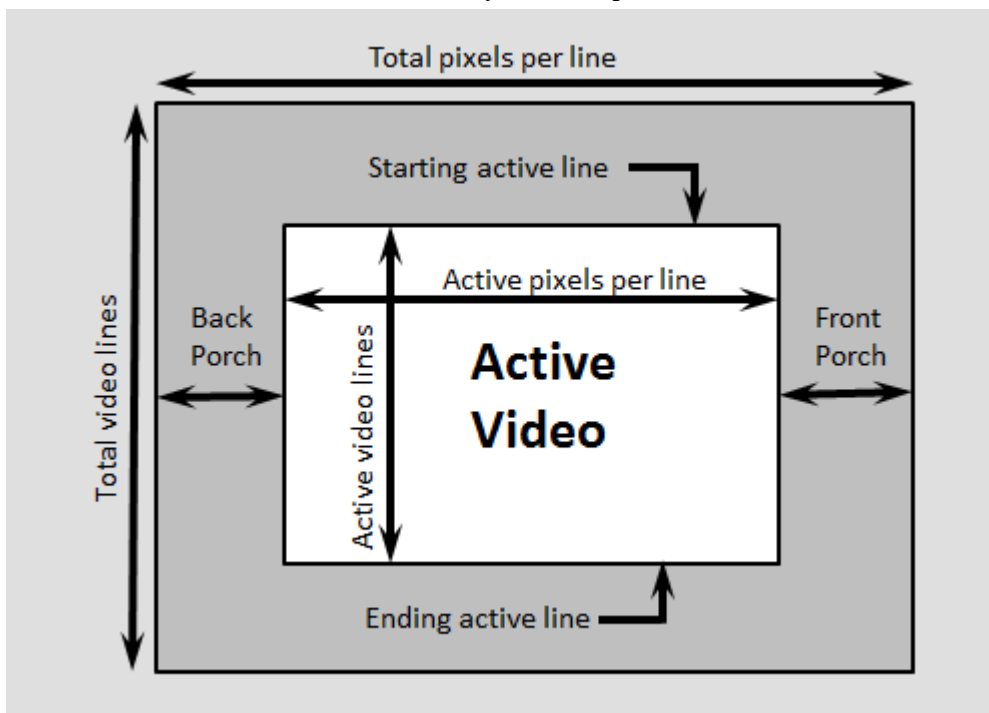


Simulating serial video in the MATLAB interpreted language can be time-consuming. Once you have debugged the design with a small image size, use MEX code generation to accelerate testing with larger images. See “Accelerate a MATLAB Design With MATLAB Coder”.

Serialize Data

The `visionhdl.FrameToPixels` System object converts framed video to a pixel stream and control structure. This object provides input for a function targeted for HDL code generation, but it does not itself support HDL code generation.

To simulate with a standard video format, choose a predefined video padding format to match your input source. To simulate with a custom-sized image, choose dimensions of inactive regions to surround the image. This tutorial uses a custom image. The properties of the `visionhdl.FrameToPixels` object correspond to the dimensions in the diagram.



Create a `visionhdl.FrameToPixels` object and set the image properties. The image is an intensity image with a scalar value representing each pixel, therefore set `NumComponents` property to 1. This tutorial pads the thumbnail image with 5 inactive lines above and below, and 10 inactive pixels on the front and back of each line.

```
frm2pix = visionhdl.FrameToPixels(...
    'NumComponents', 1, ...
```



```

'VideoFormat', 'custom', ...
'ActivePixelsPerLine', imActivePixels, ...
'ActiveVideoLines', imActiveLines, ...
'TotalPixelsPerLine', imActivePixels+20, ...
'TotalVideoLines', imActiveLines+10, ...
'StartingActiveLine', 6, ...
'FrontPorch', 10);

```

Use the `getparamfromfrm2pix` function to get useful image dimensions from the serializer object. This syntax discards the first two returned values, and keeps only the total number of pixels in the padded frame.

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

Call the object to convert the image into a vector of pixels and a vector of control signals.

Note: This syntax runs only in R2016b or later. If you are using an earlier release, replace each call of an object with the equivalent `step` syntax. For example, replace `myObject(x)` with `step(myObject,x)`.

```
[pixel,ctrl] = frm2pix(inputIm);
```

Preallocate the output vectors for a more efficient simulation.

```
pixelOut = zeros(numPixelsPerFrame,1,'uint8');
ctrlOut = repmat(pixelcontrolstruct,numPixelsPerFrame,1);
```

Design HDL-Compatible Model

Select an image processing object from the `visionhdl` library. This tutorial uses `visionhdl.ImageFilter`.

Construct a function containing a persistent instance of this object. The function processes a single pixel by executing one call to the `step` method of the object.

The `ctrlIn` and `ctrlOut` arguments of the `step` method are structures that contain five control signals. The signals indicate the validity of each pixel and the location of each pixel in the frame.

Set the filter coefficients of the `visionhdl.ImageFilter` to perform a 2×2 blur operation.

For this tutorial, you do not need to change the `LineBufferSize` property of the filter object. This parameter does not affect simulation speed, so it does not need to be modified when simulating with a small test image. When choosing `LineBufferSize`, select a power of two that accommodates the active line size of the largest required frame format. The default value, 2048, accommodates 1080p video format.

```
function [pixOut,ctrlOut] = HDLTargetedDesign(pixIn,ctrlIn)

    persistent filt2d
    if isempty(filt2d)
        filt2d = visionhdl.ImageFilter(...
            'Coefficients',ones(2,2)/4,...
            'CoefficientsDataType','Custom',...
            'CustomCoefficientsDataType',numericType(0,1,2),...
            'PaddingMethod','Symmetric');
    end

    [pixOut,ctrlOut] = filt2d(pixIn,ctrlIn);
end
```

Call the function once for each pixel in the padded frame, which is represented by the pixel vector.

```
for p = 1:numPixelsPerFrame
    [pixelOut(p),ctrlOut(p)] = HDLTargetedDesign(pixel(p),ctrl(p));
end
```

Deserialize Filtered Pixel Stream

The `visionhdl.PixelsToFrame` System object converts a pixel stream to frame-based video. Use this object to deserialize the filtered data from `visionhdl.ImageFilter`. Set the image dimension properties to match the test image.

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

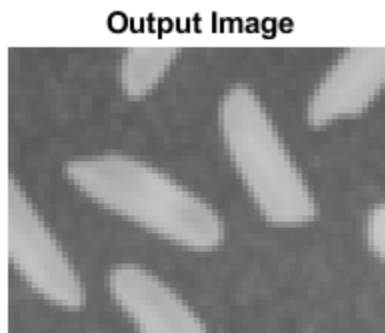
Call the object to convert the output of the HDL-targeted function to a matrix.

```
[outputIm,validIm] = pix2frm(pixelOut,ctrlOut);
```

Display Results

Use the `imshow` function to display the result of the operation.

```
if validIm
    figure
    imshow(outputIm, 'InitialMagnification', 300)
    title 'Output Image'
end
```



Compare to Behavioral Model

If you have a behavioral model of the design, you can compare the output frames visually or mathematically. For filtering, you can compare `visionhdl.ImageFilter` with the `imfilter` function in Image Processing Toolbox™. The `imfilter` function operates on the frame as a matrix and return a modified frame as a matrix. You can compare this matrix with the matrix output of the `pix2frm` object.

To avoid dependency on a Image Processing Toolbox license, this tutorial does not perform a compare.

HDL Code Generation

Once your design is working in simulation, use HDL Coder to generate HDL code for the `HDLTargetedDesign` function. See “Generate HDL Code From MATLAB”.

See Also

Related Examples

- “Pixel-Streaming Design in MATLAB”
- “Accelerate a Pixel-Streaming Design Using MATLAB Coder”

Configure the Simulink Environment for HDL Video Processing

In this section...

“About Simulink Model Templates” on page 1-17

“Create Model Using Vision HDL Toolbox Model Template” on page 1-17

“Vision HDL Toolbox Model Template” on page 1-18


About Simulink Model Templates

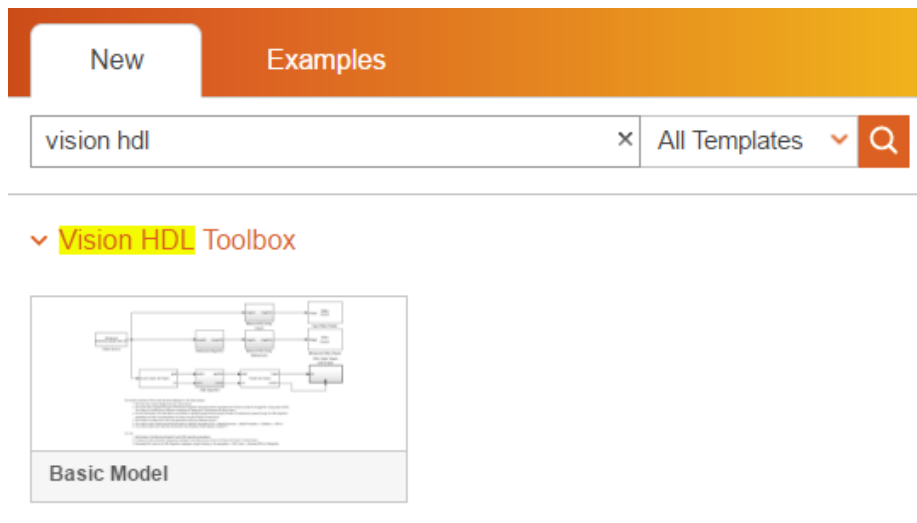
Simulink model templates provide common configuration settings and best practices for new models. Instead of the default canvas of a new model, select a template model to help you get started.

For more information on Simulink model templates, see “Create a Model” (Simulink).

Create Model Using Vision HDL Toolbox Model Template

To use the Vision HDL Toolbox model template:

- 1 Click the Simulink button, , or type `simulink` at the MATLAB command prompt.
- 2 On the Simulink start page, find the Vision HDL Toolbox section, and click the **Basic Model** template.



A new model, with the template contents and settings, opens in the Simulink Editor. Click **File > Save as** to save the model.

You can also create a new model from the template on the command line.

```
new_system my_visionhdl_model FromTemplate visionhdl_basic.sltx
open_system my_visionhdl_model
```

Vision HDL Toolbox Model Template

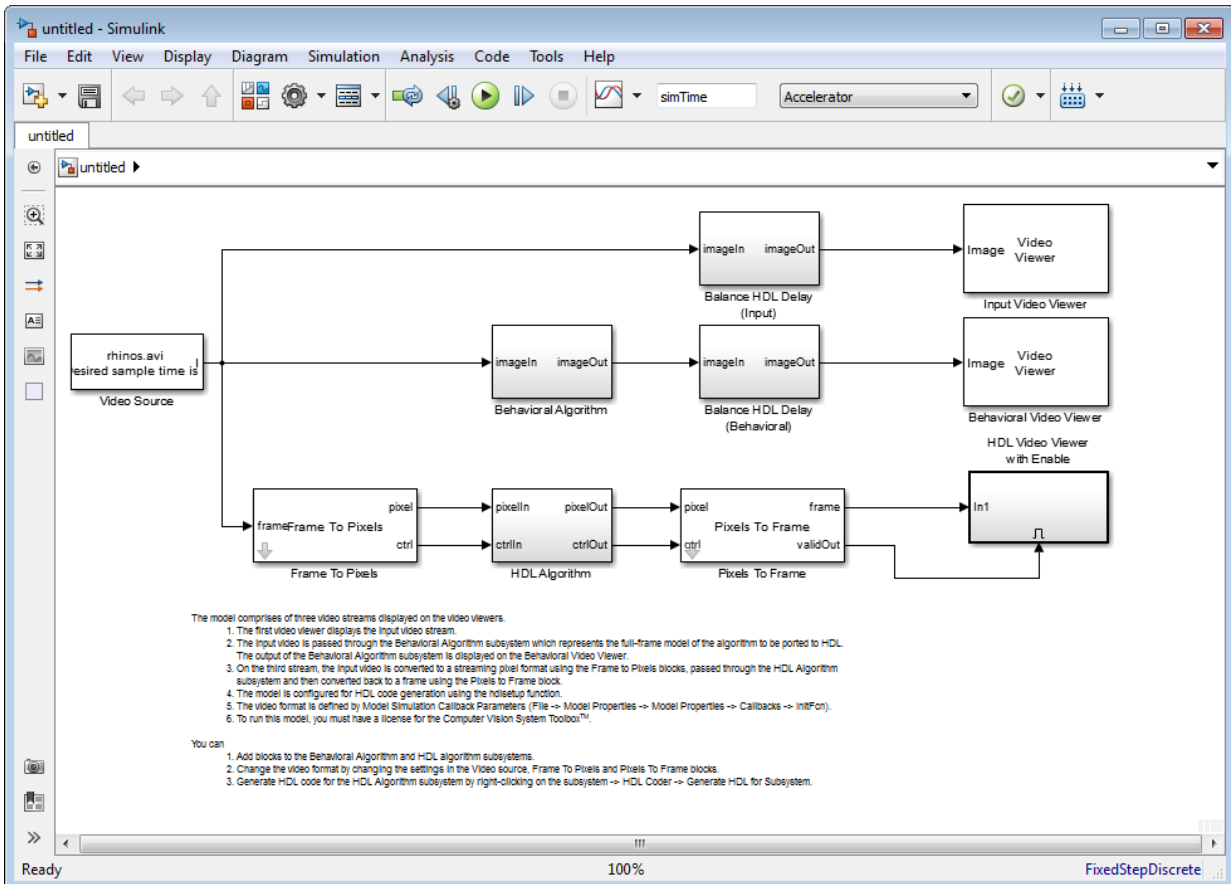
Basic Model Template

The Vision HDL Toolbox **Basic Model** template includes the following features:

- Blocks to convert framed video data to a pixel stream, and to convert the output pixel stream back to full-frame video
- An empty behavioral model subsystem
- An empty HDL-targeted subsystem

- Display blocks to compare the results of the two subsystems
- Delay blocks on the input and behavioral model data paths. These delays match the one-frame delay introduced by the Pixels To Frame conversion on the HDL model data path.

This template also configures the model for HDL code generation.



This template uses the Video Source and Video Viewer blocks from Computer Vision System Toolbox.

Due to serial processing, Vision HDL Toolbox simulation can be time-consuming for large images. You can work around this limitation by designing and debugging with a small

image, and then increasing the size before final testing and HDL code generation. The pixel stream control signals allow most blocks, except for those for frame and pixel conversion, to be independent of image size. To change image size, modify the Frame To Pixels and Pixels To Frame block parameters only. To simplify a size change, use variables for custom-size image dimensions. This template uses the standard 240p format and also provides image dimension variables in the callback function, `InitFcn`. These variables control the sample time on the Video Source and the simulation stop time. To view or edit this function, click **File > Model Properties > Model Properties**, select the **Callbacks** tab, and then click `InitFcn*`.

This template includes the following features that assist with HDL code generation:

- Configures Solver settings equivalent to calling `hdlsetup`
- Displays data rates and data types in the Model Editor
- Creates an instance of `pixelcontrolbus` in the workspace (in `InitFcn`)
- Enables `fileIO` mode when generating an HDL test bench

See Also

Related Examples

- “Design Video Processing Algorithms for HDL in Simulink” on page 1-3