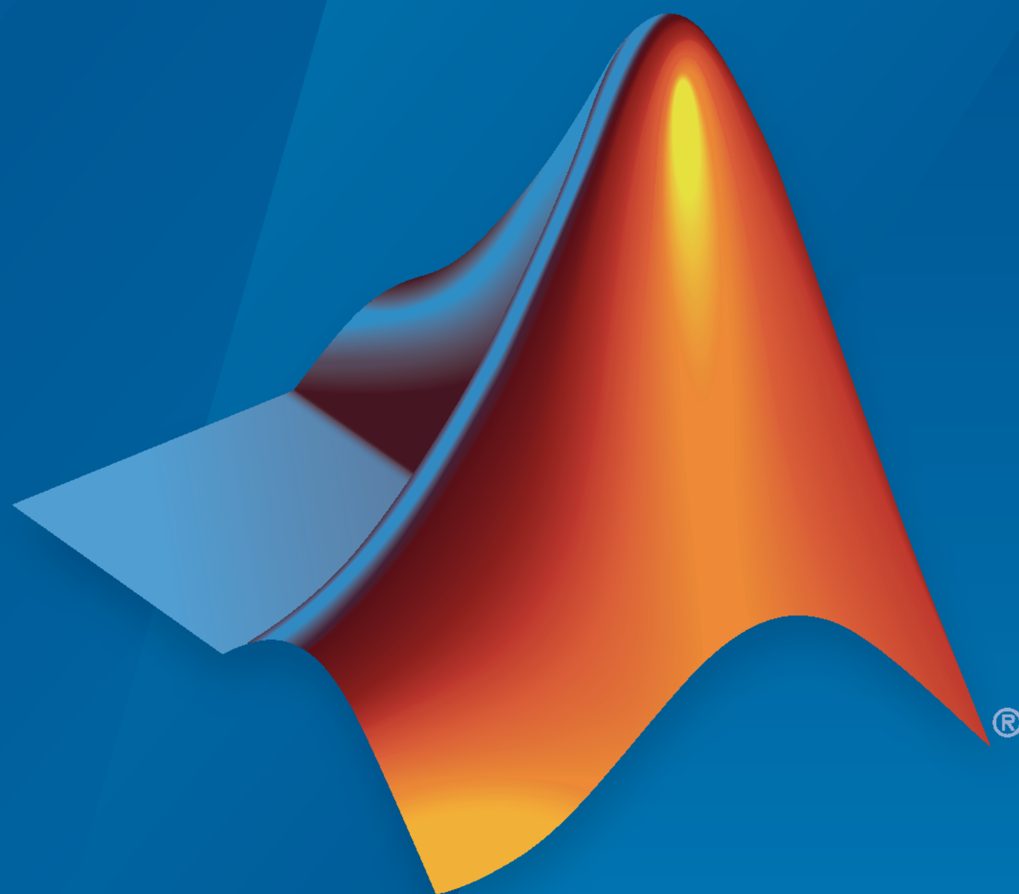
Vision HDL Toolbox™

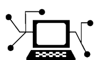
Getting Started Guide



MATLAB®

R2020a

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.
1 Apple Hill Drive
Natick, MA 01760-2098

Vision HDL Toolbox™ Getting Started Guide

© COPYRIGHT 2015–2020 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)
March 2018	Online only	Revised for Version 1.6 (Release 2018a)
September 2018	Online only	Revised for Version 1.7 (Release 2018b)
March 2019	Online only	Revised for Version 1.8 (Release 2019a)
September 2019	Online only	Revised for Version 2.0 (Release 2019b)
March 2020	Online only	Revised for Version 2.1 (Release 2020a)

Contents

1 Vision HDL Toolbox Getting Started

Vision HDL Toolbox Product Description **1-2**

Design Video Processing Algorithms for HDL in Simulink **1-3**

Design a Hardware-Targeted Image Filter in MATLAB **1-10**

MATLAB Vision Algorithm to Simulink Hardware-Targeted Model Workflow **1-15**

Configure the Simulink Environment for HDL Video Processing **1-20**

About Simulink Model Templates **1-20**

Create Model Using Vision HDL Toolbox Model Template **1-20**

Vision HDL Toolbox Model Template **1-21**

Accelerate a Pixel-Streaming Design Using MATLAB Coder **1-23**

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 a Hardware-Targeted Image Filter in MATLAB” on page 1-10
- “MATLAB Vision Algorithm to Simulink Hardware-Targeted Model Workflow” on page 1-15
- “Configure the Simulink Environment for HDL Video Processing” on page 1-20
- “Accelerate a Pixel-Streaming Design Using MATLAB Coder” on page 1-23

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. 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 is FPGA-proven for frame sizes up to 8k resolution and for high frame rate (HFR) video.

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

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 Toolbox™ blocks.

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

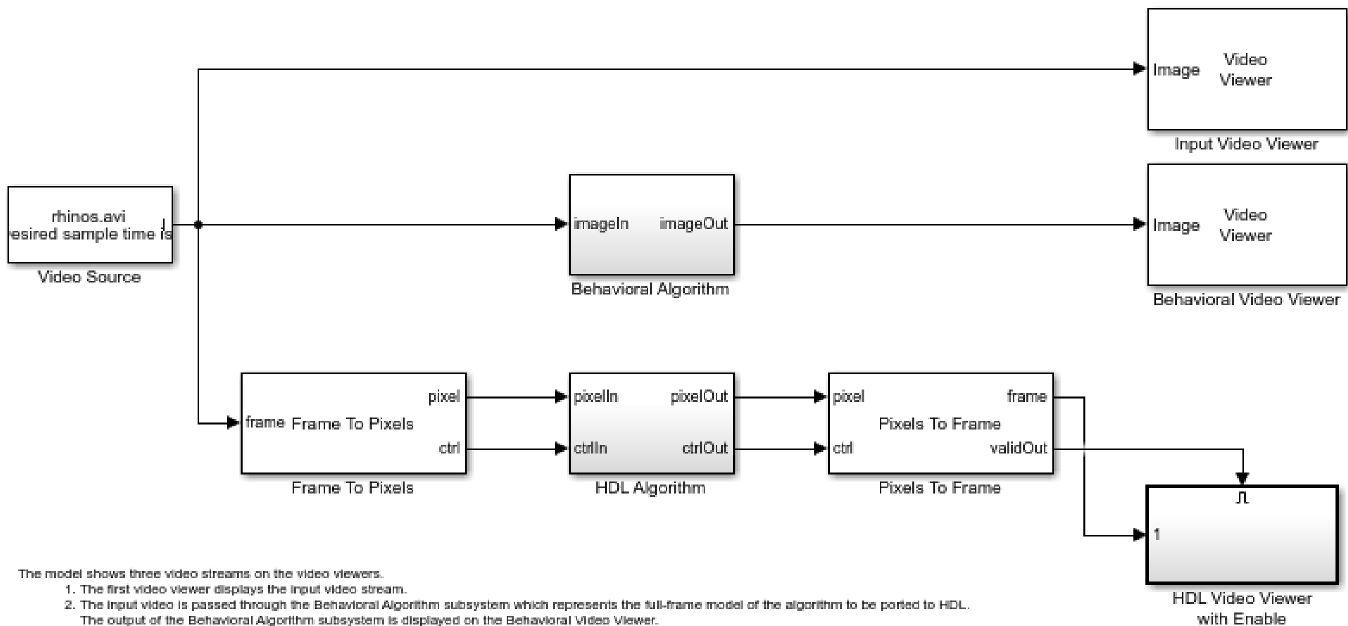
- **Streaming pixel interface**: Blocks in Vision HDL Toolbox use a streaming pixel interface. Serial processing is efficient for hardware designs, because less memory is required to store pixel data for computation. The serial interface allows the block to operate independently of image size and format 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.

Open Model Template

This tutorial uses a Simulink model template to get started.

Click the Simulink button, or type `simulink` at the MATLAB® command prompt. 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.

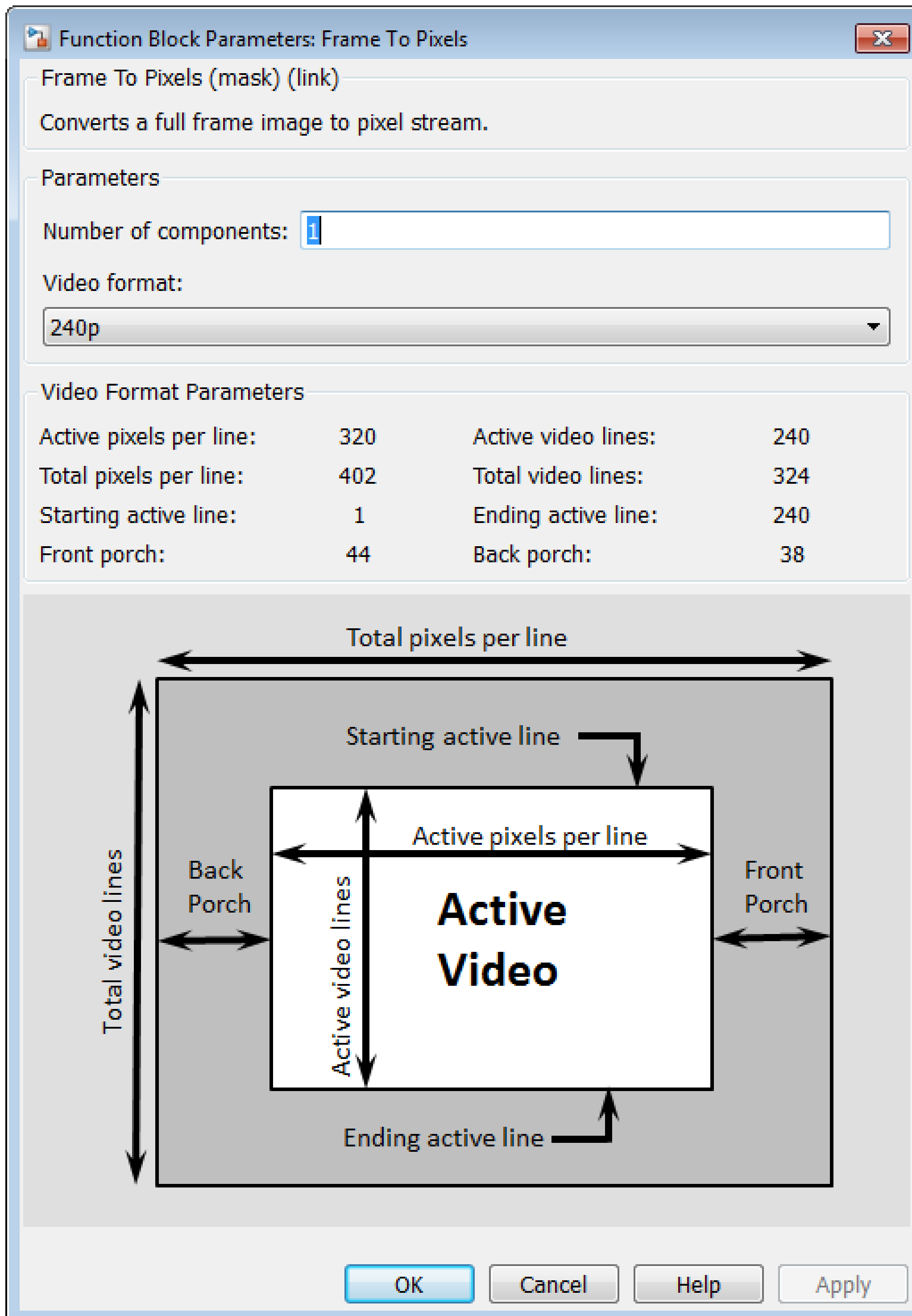
Serialize Data

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 `visionhdllib` 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)`.

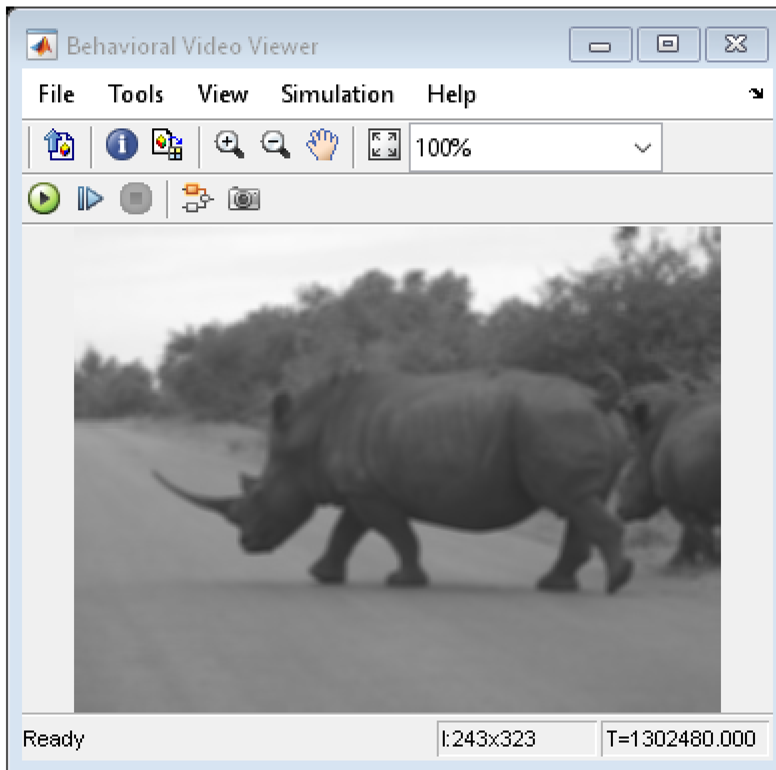
Deserialize Filtered Pixel Stream

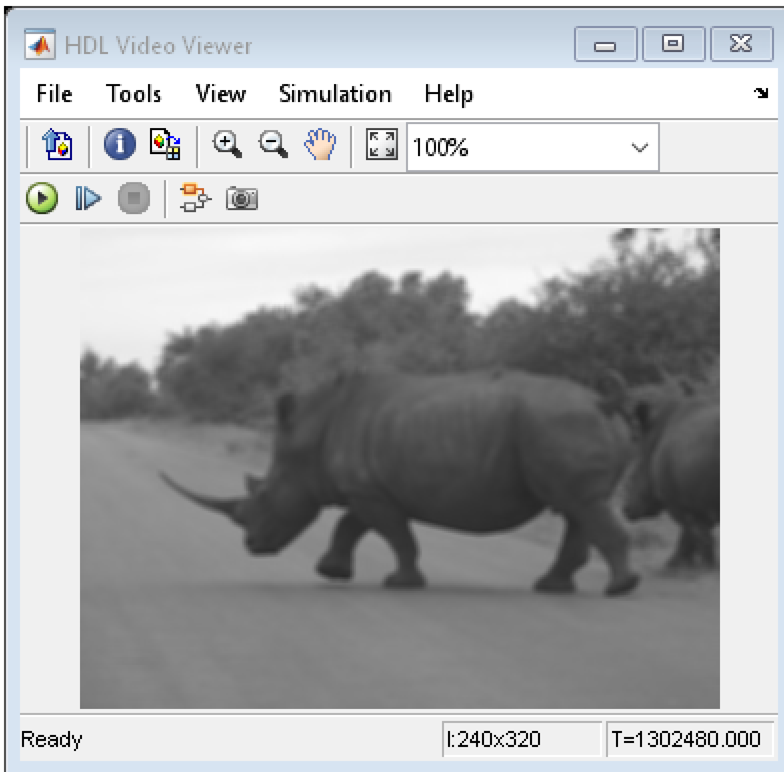
Use the Pixels To Frame block included in the template to deserialize the data for display.

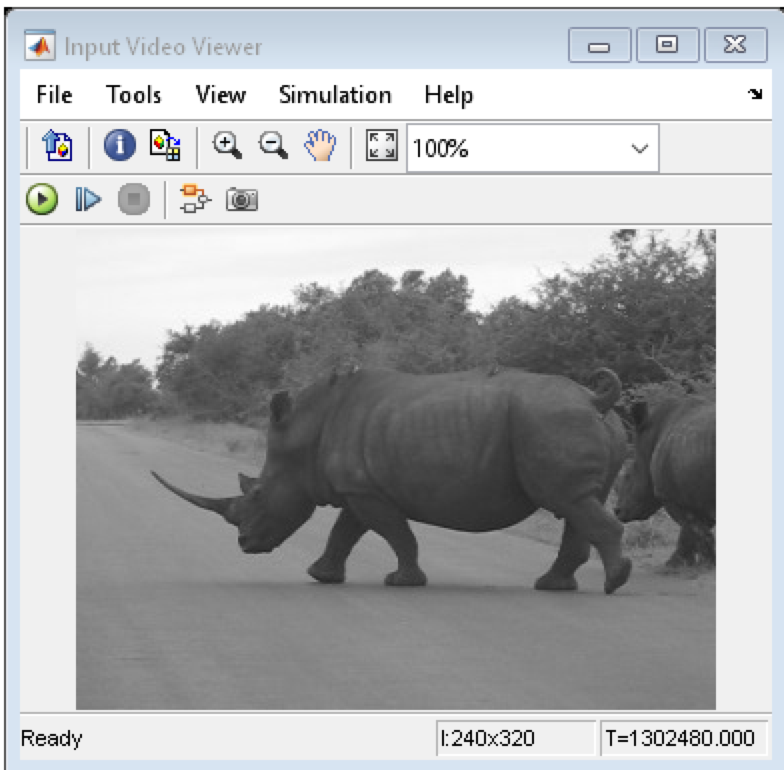
Open the Pixels To Frame block. Set the image dimension properties to match the input video and the settings you specified in the Frame To Pixels block. For this tutorial, the **Number of components** is set to 1 and the **Video format** is set to 240p. The block converts the stream of output pixels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




HDL Video Viewer
File
Tools
View
Simulation
Help
100%
Ready
I:240x320
T=1302480.000


Input Video Viewer
File
Tools
View
Simulation
Help
100%
Ready
I:240x320
T=1302480.000

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 1-20

Design a Hardware-Targeted Image Filter 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.

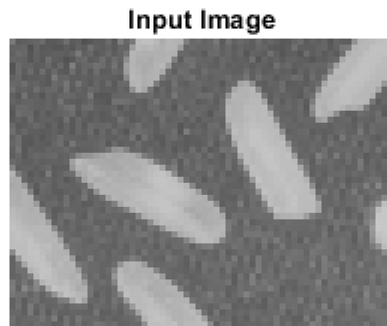
Import Data

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.

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 Pixel-Streaming Design Using MATLAB Coder” on page 1-23.

```
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'
```

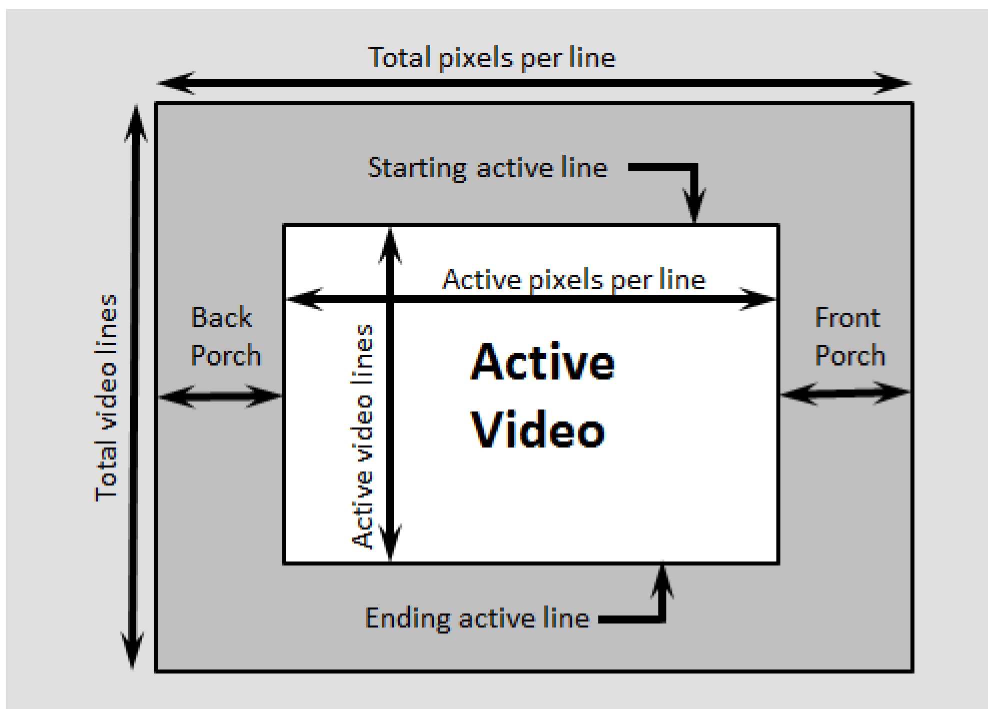
```
origImSize =

   256   256
```



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.

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. 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)`.

```
frm2pix = visionhdl.FrameToPixels(...
         'NumComponents',1,...
         'VideoFormat','custom',...
         'ActivePixelsPerLine',imActivePixels,...
         'ActiveVideoLines',imActiveLines,...
         'TotalPixelsPerLine',imActivePixels+20,...
         'TotalVideoLines',imActiveLines+10,...
         'StartingActiveLine',6,...
         'FrontPorch',10);

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

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 object.

The `ctrlIn` and `ctrlOut` arguments of the object 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
```

Preallocate the output vectors for a more efficient simulation. Then, call the function once for each pixel in the padded frame, which is represented by the `pixel` vector.

```
pixelOut = zeros(numPixelsPerFrame,1,'uint8');
ctrlOut  = repmat(pixelcontrolstruct,numPixelsPerFrame,1);
 for p = 1:numPixelsPerFrame
    [pixelOut(p),ctrlOut(p)] = HDLTargetedDesign(pixel(p),ctrl(p));
 end
```

Deserialize Filtered Pixel Stream

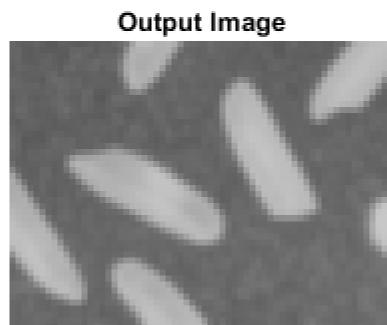
The `visionhdl.PixelsToFrame` System object converts a pixel stream to frame-based video. Use this object to deserialize the filtered data from `visionhdl.ImageFilter`. Set the image dimension properties to match the test image. Call the object to convert the output of the HDL-targeted function to a matrix.

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

[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” on page 1-23

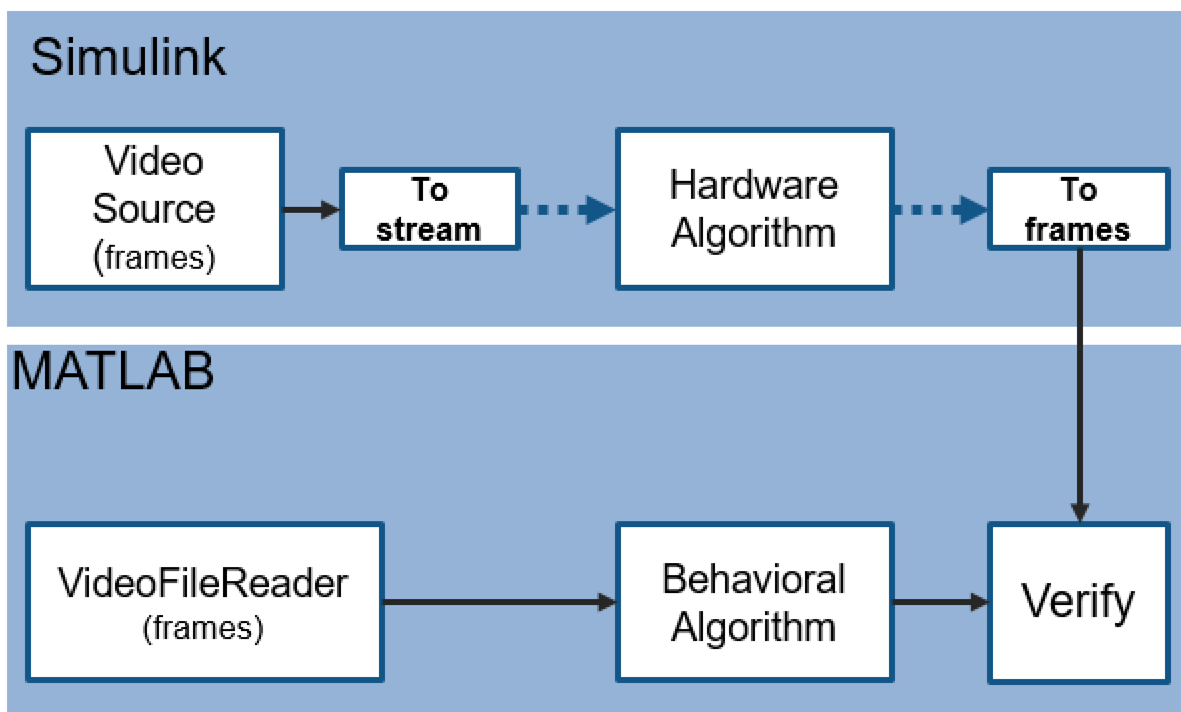
MATLAB Vision Algorithm to Simulink Hardware-Targeted Model Workflow

This example shows how to create a hardware-targeted design in Simulink® that implements the same behavior as a MATLAB® reference design.

Workflow

Image Processing Toolbox™ and Computer Vision Toolbox™ functions operate on framed, floating-point and integer data and provide excellent behavioral references. Hardware designs must use streaming Boolean or fixed-point data.

This example shows how to perform a framed image processing operation in MATLAB, and then implement the same operation in a Simulink model using streaming data. The Simulink model converts the input video to a pixel stream for hardware-friendly design. The same data is applied to both the hardware algorithm in Simulink and the behavioral algorithm in MATLAB. The Simulink model converts the output pixel stream to frames and exports those frames to MATLAB for comparison against the behavioral results.



The MATLAB portion of this example loads the input video, runs the behavioral code, runs the Simulink model to import video frames and export modified video frames, and compares the MATLAB behavioral results with the Simulink output frames.

Video Source

Create a video file reader object to import a video file into the MATLAB workspace. The video source file is 240p format. Create a video player object to display the input frame, Simulink filtered frame, and MATLAB reference frame.

```
videoIn = vision.VideoFileReader(...
        'Filename','rhinos.avi',...
        'ImageColorSpace','Intensity',...
        'VideoOutputDataType','uint8');

numFrm = 10;
% active frame dimensions
actPixelsPerLine = 320;
actLines = 240;
% dimensions including blanking
totalPixelsPerLine = 402;
totalLines = 324;

% viewer for results
viewer = vision.DeployableVideoPlayer(...
        'Size','Custom',...
        'CustomSize',[3*actPixelsPerLine actLines]);
```

Edge Detection and Overlay

Detect edges in the video frames, and then overlay those edges onto the original frame. The overlay computation uses an `alpha` value to mix the two pixel values. The Simulink model also uses the `edgeThreshold` and `alpha` parameters specified here.

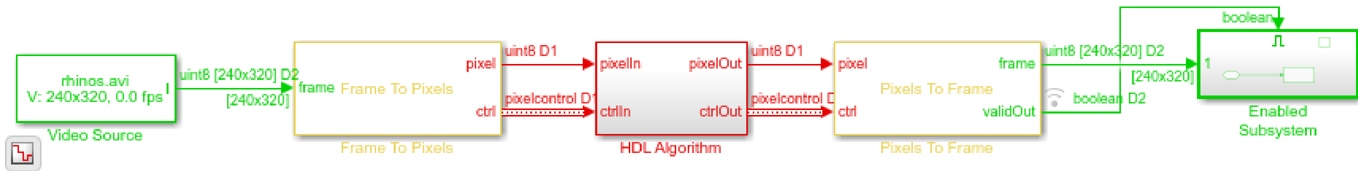
The MATLAB `edge` function interprets the threshold as a double-precision value from 0 to 1. Therefore, express the threshold as a fraction of the range of the `uint8` data type, from 0 to 255. The pixel values returned by the `edge` function are `logical` data type. To convert these pixel values to `uint8` type for overlay, multiply by 255. This scaling operation converts logical ones to 255 and logical zeros stay 0.

```
edgeThreshold = 8;
alpha = 0.75;
frmFull = uint8(zeros(actLines,actPixelsPerLine,numFrm));
frmRef = frmFull;
for f = 1:numFrm
    frmFull(:,:,f) = videoIn();
    edges = edge(frmFull(:,:,f),'Sobel',edgeThreshold/255,'nothinning');
    edges8 = 255*uint8(edges)*(1-alpha);
    frmRef(:,:,f) = alpha*frmFull(:,:,f) + edges8;
    viewer([edges edges8 frmRef(:,:,f)]);
end
```

Set Up for Simulink Simulation

The Simulink model loads the input video into the model using a Video Source block. Configure the sample time of the model using the `totPixPerFrame` variable. This value includes the inactive pixel regions around the 240-by-320 frames. The Video Source sample time is 1 time step per frame, and the rate in the streaming pixel sections of the model is 1/ `totPixPerFrame` . Set the length of the simulation with the `simTime` variable.

```
totPixPerFrame = totalPixelsPerLine*totalLines;
simTime = (numFrm+1)*totPixPerFrame;

modelname = 'VerifySLDesignAgainstMLReference';
open_system(modelname);
set_param(modelname,'SampleTimeColors','on');
set_param(modelname,'SimulationCommand','Update');
set_param(modelname,'Open','on');
```



Hardware-Targeted Algorithm

The HDL Algorithm subsystem is designed to support HDL code generation.

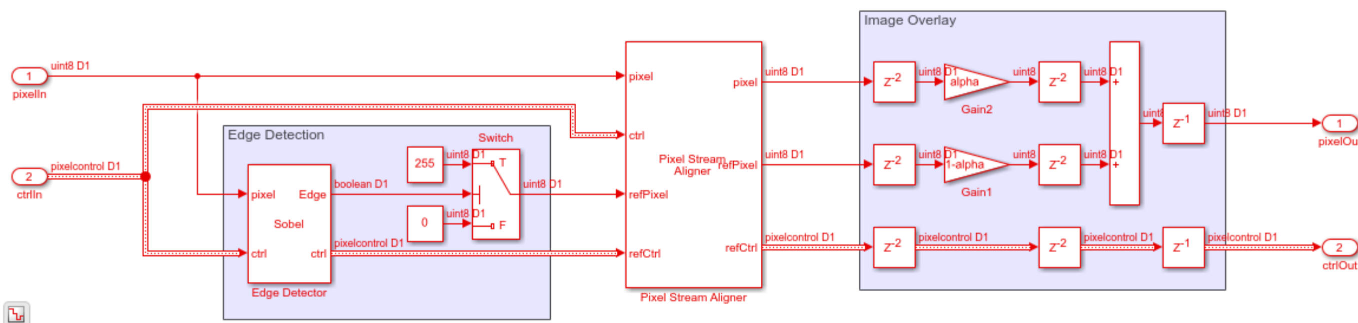
The subsystem uses the Edge Detector block to find edges. The output of the block is a stream of `boolean` pixel values. The model scales these values to `uint8` data type values for overlay.

The block returns the pixel stream of detected edges after several lines of latency, due to internal line buffers and filter logic. Before performing overlay, the model must delay the input stream to match the edge stream. The Pixel Stream Aligner block performs this alignment using the control signals of the output edge stream as a reference. This block stores the input stream in a FIFO until the detected edges are available.

The Image Overlay subsystem scales both streams by the `alpha` ratio and adds them together. With hardware implementation in mind, the Image Overlay subsystem includes pipeline stages around each multiplier and after the adder.

For more details of this edge detector design, see the “Edge Detection and Image Overlay” example.

```
open_system([modelname '/HDL Algorithm']);
```



Run Simulink Model

Run the Simulink model to return ten frames overlaid with the detected edges.

```
sim('VerifySLDesignAgainstMLReference');
```

Compare Simulink Results with MATLAB Results

Compare each video frame returned from Simulink with the result returned by the MATLAB behavioral code. The images look very similar but have small pixel value differences due to overlay mixing. The MATLAB overlay mixing is done using floating-point values, and the Simulink overlay mixing is done using fixed-point values. This comparison counts pixels in each frame whose values differ by more than 2 and calculates the peak-signal-to-noise ratio (PSNR) between the frames. To view the detailed differences at each frame, uncomment the last two lines in the loop.

```
for f = 1:numFrm
   frmResult = frmOut.signals.values(:,:,f);
   viewer([frmFull(:,:,f) frmResult frmRef(:,:,f)]);
   diff = frmRef(:,:,f) - frmResult;
   errcnt = sum(diff(:) > 2);
   noisecheck = psnr(frmRef(:,:,f),frmResult);
   fprintf(['\nFrame #%d has %d pixels that differ from behavioral result (by more than 2). PSNR
   %bar3(diff);
   %viewer([frmResult frmRef(:,:,f) diff]);
end
```

```
Frame #1 has 2 pixels that differ from behavioral result (by more than 2). PSNR = 48.33

Frame #2 has 1 pixels that differ from behavioral result (by more than 2). PSNR = 48.72

Frame #3 has 1 pixels that differ from behavioral result (by more than 2). PSNR = 48.80

Frame #4 has 2 pixels that differ from behavioral result (by more than 2). PSNR = 48.66

Frame #5 has 2 pixels that differ from behavioral result (by more than 2). PSNR = 48.70

Frame #6 has 4 pixels that differ from behavioral result (by more than 2). PSNR = 48.27

Frame #7 has 2 pixels that differ from behavioral result (by more than 2). PSNR = 48.88

Frame #8 has 3 pixels that differ from behavioral result (by more than 2). PSNR = 48.58

Frame #9 has 3 pixels that differ from behavioral result (by more than 2). PSNR = 48.55

Frame #10 has 3 pixels that differ from behavioral result (by more than 2). PSNR = 48.53
```

Generate HDL Code and Verify Its Behavior

Once your design is working in simulation, you can use HDL Coder™ to generate HDL code and a test bench for the HDL Algorithm subsystem.

```
makehdl([modelname '/HDL Algorithm'])   % Generate HDL code
makehdltb([modelname '/HDL Algorithm']) % Generate HDL Test bench
```

See Also
Edge Detector | Pixel Stream Aligner

More About

- “Streaming Pixel Interface”
- “Configure the Simulink Environment for HDL Video Processing” on page 1-20
- “Edge Detection and Image Overlay”

Configure the Simulink Environment for HDL Video Processing

In this section...
“About Simulink Model Templates” on page 1-20
“Create Model Using Vision HDL Toolbox Model Template” on page 1-20
“Vision HDL Toolbox Model Template” on page 1-21

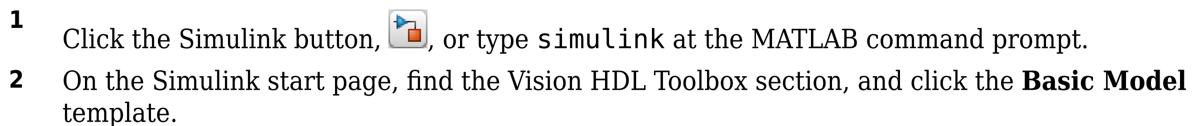
About Simulink Model Templates

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.

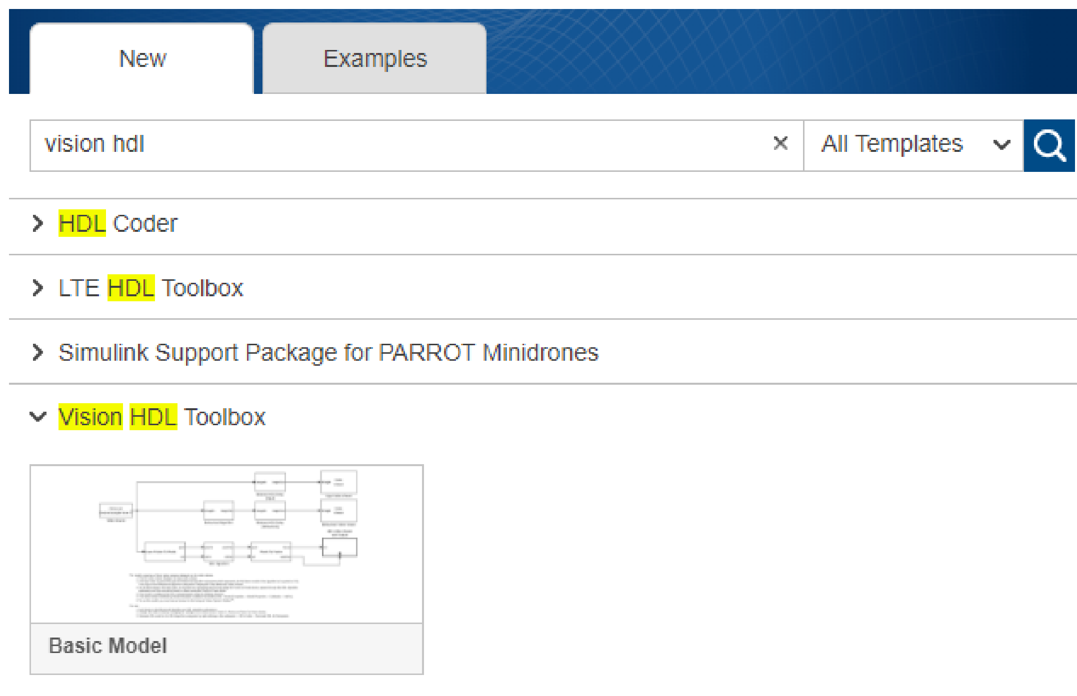
For more information on Simulink model templates, see “Build and Edit a Model Interactively” (Simulink).

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 **Save** 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
```

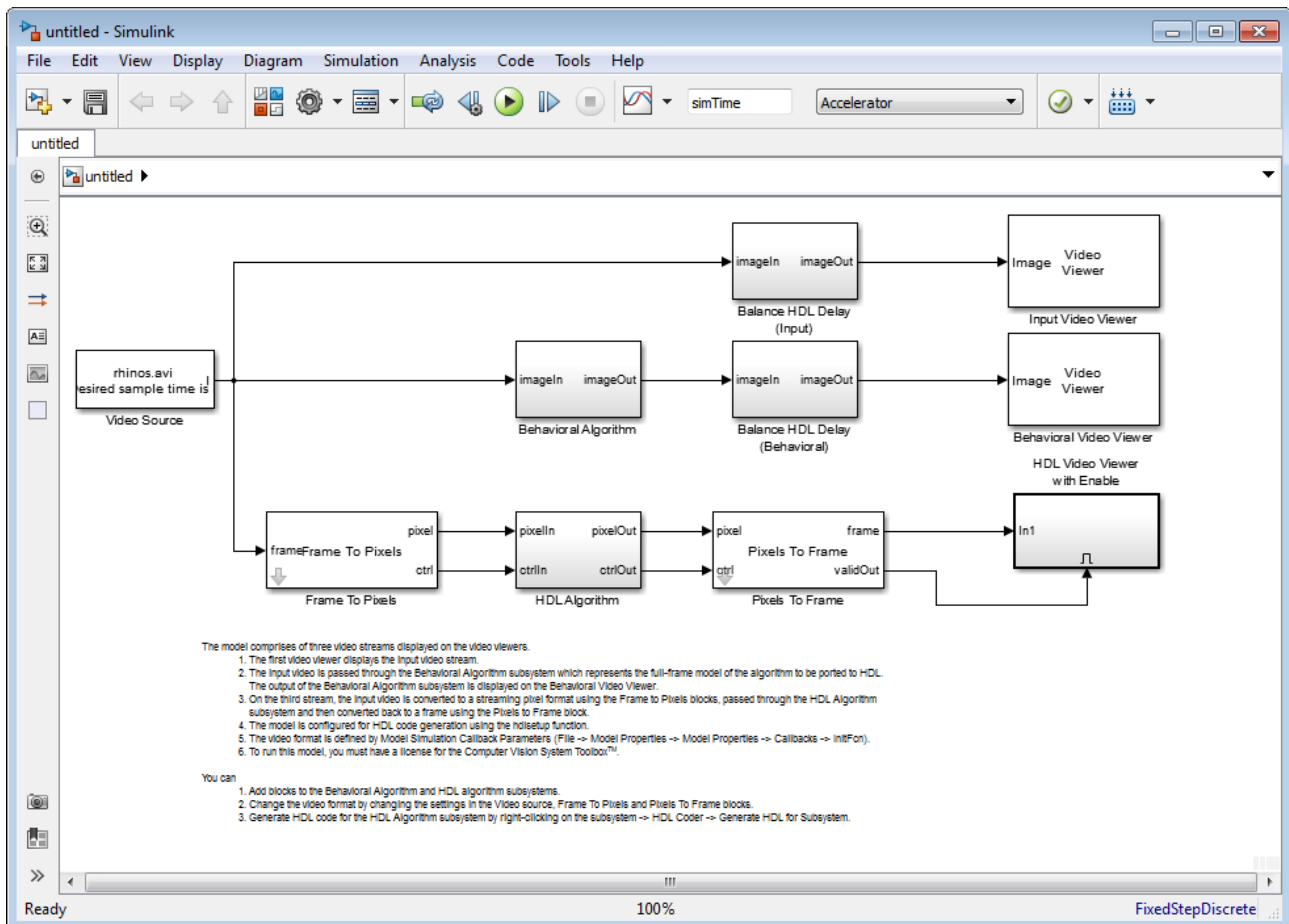
Vision HDL Toolbox Model Template

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 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, on the **Modeling** tab, expand **Model Settings** and click **Model Properties**, select the **Callbacks** tab, and then click `InitFcn*`.

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

- Configures Solver and HDL Code Generation settings equivalent to calling `visionhdlsetup`
- 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

Accelerate a Pixel-Streaming Design Using MATLAB Coder

This example demonstrates a workflow for accelerating a pixel-stream video processing algorithm using MATLAB Coder™ and generating HDL code from the design. You must have a MATLAB Coder license to run this example.

Acceleration with MATLAB Coder enables you to simulate large frame sizes, such as 1080p video, at practical speeds. Use this acceleration workflow after you have debugged the algorithm using a small frame size. Testing a design with a small image is demonstrated in the "Pixel-Streaming Design in MATLAB" example.

How MATLAB Coder Works

MATLAB Coder generates C code from MATLAB® code. Code generation accelerates simulation by locking-down the sizes and data types of variables. This process removes the overhead of the interpreted language checking for size and data type in every line of code. This example compiles both the test bench file `DesignAccelerationHDLTestBench.m` and the design file `DesignAccelerationHDLDesign.m` into a MEX function, and uses the resulting MEX file to speed up the simulation.

The directive (or pragma) **%#codegen** beneath the function signature indicates that you intend to generate code for the MATLAB algorithm. Adding this directive instructs the MATLAB code analyzer to help you diagnose and fix violations that would result in errors during code generation. The directive **%#codegen** does not affect interpreted simulation.

Best Practices

Debugging simulations with large frame sizes is impractical in interpreted mode due to long simulation time. However, debugging a MEX simulation is challenging due to lack of debug access into the code.

To avoid these scenarios, a best practice is to develop and verify the algorithm and test bench using a thumbnail frame size. In most cases, the HDL-targeted design can be implemented with no dependence on frame size. Once you are confident that the design and test bench are working correctly, then increase the frame size in the test bench, and use MATLAB Coder to accelerate the simulation. To increase the frame size, test bench only requires minor changes, as you can see by comparing `DesignAccelerationHDLTestBench.m` with the `PixelStreamingDesignHDLTestBench.m` in "Pixel-Streaming Design in MATLAB".

Test Bench

In the test bench `DesignAccelerationHDLTestBench.m`, the **videoIn** object reads each frame from a video source, and the **scaler** object interpolates this frame from 240p to 1080p. This 1080p image is passed to the **frm2pix** object, which converts the full image frame to a stream of pixels and control structures. The function `DesignAccelerationHDLDesign` is then called to process one pixel (and its associated control structure) at a time. After we process the entire pixel-stream and collect the output stream, the **pix2frm** object converts the output stream to full-frame video. The **DesignAccelerationHDLViewer** function displays the output and original images side-by-side.

The workflow above is implemented in the following lines of `DesignAccelerationHDLTestBench.m`.

```
...
for f = 1:numFrm
```

```
        frmFull = step(videoIn);       % Get a new frame
        frmIn = step(scaler,frmFull);  % Enlarge the frame

        [pixInVec,ctrlInVec] = step(frm2pix,frmIn);
        for p = 1:numPixPerFrm
            [pixOutVec(p),ctrlOutVec(p)] =...
                    visionhdlsobel_design(pixInVec(p),ctrlInVec(p));
        end
        frmOut = step(pix2frm,pixOutVec,ctrlOutVec);

        DesignAccelerationHDLViewer(actPixPerLine,actLine,[frmIn uint8(255*frmOut)]);
    end
    ...
```

The data type of frmIn is uint8 while that of frmOut, the edge detection output, is logical. Matrices of different data types cannot be concatenated, so **uint8(255*frmOut)** maps logical false and true to uint8(0) and uint8(255), respectively.

Both **frm2pix** and **pix2frm** are used to convert between full-frame and pixel-stream domains. The inner for-loop performs pixel-stream processing. The rest of the test bench performs full-frame processing (i.e., **videoIn**, **scaler**, and **viewer** inside the **DesignAccelerationHDLViewer** function).

Before the test bench terminates, frame rate is displayed to illustrate the simulation speed.

Not all functions used in the test bench support C code generation. For those that do not, such as `tic`, `toc`, `fprintf`, use **coder.extrinsic** to declare them as extrinsic functions. Extrinsic functions are excluded from MEX generation. The simulation executes them in the regular interpreted mode.

Pixel-Stream Design

The function defined in `DesignAccelerationHDLDesign.m` accepts a pixel stream and five control signals, and returns a modified pixel stream and control signals. For more information on the streaming pixel protocol used by System objects from the Vision HDL Toolbox, see “Streaming Pixel Interface”.

In this example, the function contains the Edge Detector System object.

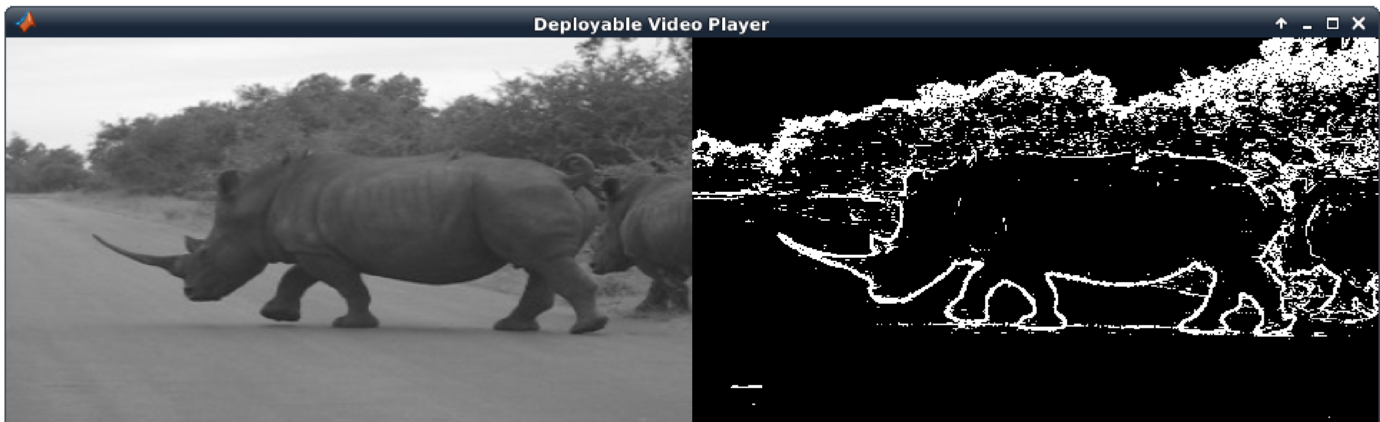
The focus of this example is the workflow, not the algorithm design itself. Therefore, the design code is quite simple. Once you are familiar with the workflow, it is straightforward to implement advanced video algorithms by taking advantage of the functionality provided by the System objects from Vision HDL Toolbox.

Create MEX File and Simulate the Design

Generate and execute the MEX file.

```
codegen('DesignAccelerationHDLTestBench');
DesignAccelerationHDLTestBench_mex;
```

```
10 frames have been processed in 20.03 seconds.
Average frame rate is 0.50 frames/second.
```



The **viewer** displays the original video on the left, and the output on the right.

HDL Code Generation

Enter the following command to create a new HDL Coder™ project in the temporary folder

```
coder -hdlcoder -new DesignAccelerationProject
```

Then, add the file `DesignAccelerationHDLDesign.m` to the project as the MATLAB Function and `DesignAccelerationHDLTestBench.m` as the MATLAB Test Bench.

Refer to “Getting Started with MATLAB to HDL Workflow” (HDL Coder) for a tutorial on creating and populating MATLAB HDL Coder projects.

Launch the Workflow Advisor. In the Workflow Advisor, right-click the 'Code Generation' step. Choose the option 'Run to selected task' to run all the steps from the beginning through HDL code generation.

Examine the generated HDL code by clicking the links in the log window.