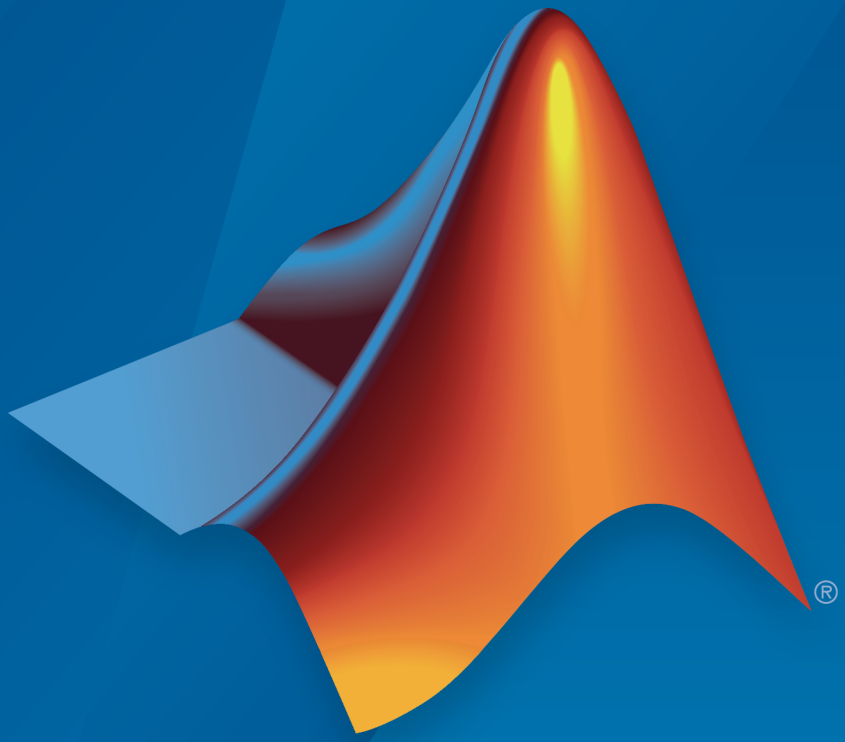
Vision HDL Toolbox™

User's Guide



MATLAB®

R2015a

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™ User's Guide

Revision History

March 2015	Online only	New for Version 1.0 (Release R2015a)

Contents

1 Streaming Pixel Interface

2 Video Interfaces and Design Best Practices

3 Prototyping

1

Streaming Pixel Interface

Streaming Pixel Interface

In this section...

What Is a Streaming Pixel Interface?

In hardware, processing an entire frame of video at one time has a high cost in memory and area. To save resources, serial processing is preferable in HDL designs. Vision HDL Toolbox blocks and System objects operate on a pixel, line, or neighborhood rather than a frame. The blocks and objects accept and generate video data as a serial stream of pixel data and control signals. The control signals indicate the relative location of each pixel within the image or video frame. The protocol mimics the timing of a video system, including inactive intervals between frames. Each block or object operates without full knowledge of the image format, and can tolerate imperfect timing of lines and frames.

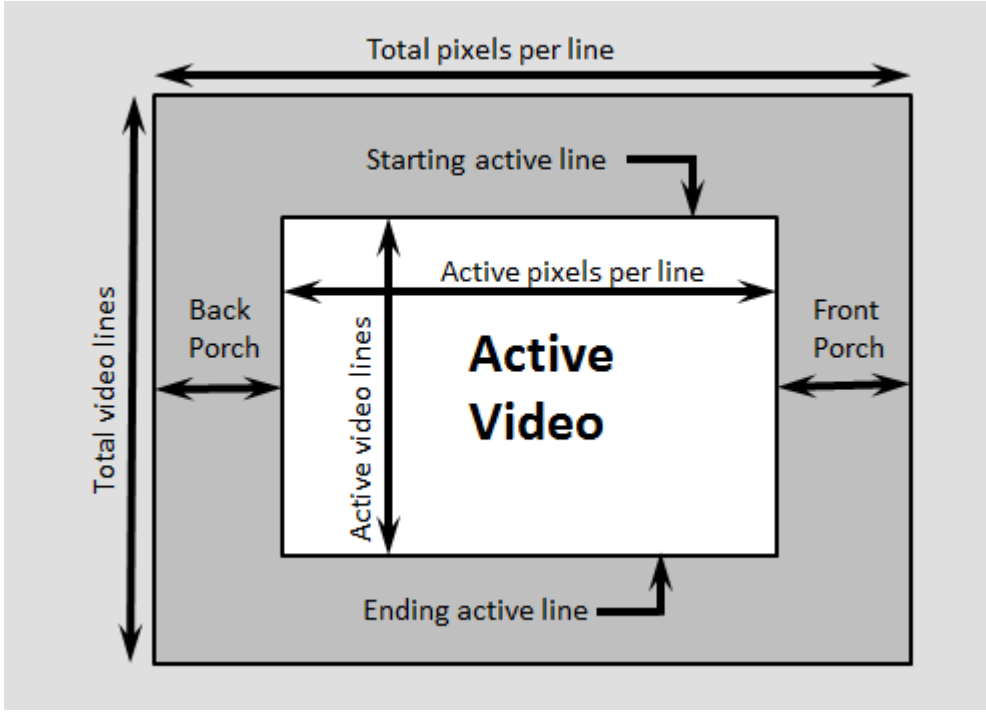
How Does a Streaming Pixel Interface Work?

Video capture systems scan video signals from left to right and from top to bottom. As these systems scan, they generate inactive intervals between lines and frames of active video.

The *horizontal blanking* interval is made up of the inactive cycles between the end of one line and the beginning of the next line. This interval is often split into two parts: the *front porch* and the *back porch*. These terms come from the synchronize pulse between lines in analog video waveforms. The *front porch* is the number of samples between the end of the active line and the synchronize pulse. The *back porch* is the number of samples between the synchronize pulse and the start of the active line.

The *vertical blanking* interval is made up of the inactive cycles between the *ending active line* of one frame and the *starting active line* of the next frame.

The scanning pattern requires start and end signals for both horizontal and vertical directions. The Vision HDL Toolbox streaming pixel protocol includes the blanking intervals, and allows you to configure the size of the active and inactive frame.



Why Use a Streaming Pixel Interface?

Format Independence

The blocks and objects using this interface do not need a configuration option for the exact image size or the size of the inactive regions. In addition, if you change the image format for your design, you do not need to update each block or object. Instead, update the image parameters once at the serialization step. Some blocks and objects still require a line buffer size parameter to allocate memory resources.

By isolating the image format details, you can develop a design using a small image for faster simulation. Then once the design is correct, update to the actual image size.

Error Tolerance

Video can come from various sources such as cameras, tape storage, digital storage, or switching and insertion gear. These sources can introduce timing problems. Human vision cannot detect small variance in video signals, so the timing for a video system does not need to be perfect. Therefore, video processing blocks must tolerate variable timing of lines and frames.

By using a streaming pixel interface with control signals, each Vision HDL Toolbox block or object starts computation on a fresh segment of pixels at the start-of-line or start-of-frame signal. The computation occurs whether or not the block or object receives the end signal for the previous segment.

The protocol tolerates minor timing errors. If the number of valid and invalid cycles between start signals varies, the blocks or objects continue to operate correctly. Some Vision HDL Toolbox blocks and objects require minimum horizontal blanking regions to accommodate memory buffer operations.

Pixel Stream Conversion Using Blocks and System Objects

In Simulink®, use the Frame To Pixels block to convert framed video data to a stream of pixels and control signals that conform to this protocol. The control signals are grouped in a nonvirtual bus data type called `pixelcontrol`.

In MATLAB®, use the `visionhdl.FrameToPixels` object to convert framed video data to a stream of pixels and control signals that conform to this protocol. The control signals are grouped in a structure data type.

If your data is already in a serial format, design your own logic to generate these control signals from your existing serial control scheme.

Supported Pixel Data Types

Vision HDL Toolbox blocks and objects include ports or arguments for streaming pixel data. The blocks and objects capture one pixel at a time from the input, and produce one pixel at a time for output. Each block and object supports one or more pixel formats. The supported formats vary depending on the operation the block or object performs. This table details common video formats supported by Vision HDL Toolbox.

Type of Video	Pixel Format
Binary	Each pixel is represented by a single `boolean` or `logical` value. Used for true black-and-white video.
Grayscale	Each pixel is represented by *luma*, which is the gamma-corrected luminance value. This pixel is a single unsigned integer or fixed-point value.
Color	Each pixel has multiple unsigned integer or fixed-point values representing the color components of the pixel. Vision HDL Toolbox blocks and objects use gamma-corrected color spaces, such as R'G'B' and Y'CbCr.

Vision HDL Toolbox blocks have an input or output port, `pixel`, for the pixel data. Vision HDL Toolbox System objects expect or return an argument to the `step` method representing the pixel data. The following table describes the format of the pixel data.

Port or Argument	Description	Data Type
`pixel`	Scalar that represents binary or grayscale pixel value, or a vector of three values representing color values.	Supported data types can include: • `boolean` or `logical` • `uint` or `int` • `fixdt()` `double` and `single` data types are supported for simulation but not for HDL code generation.

Streaming Pixel Control Signals

Vision HDL Toolbox blocks and objects include ports or arguments for control signals relating to each pixel. These five control signals indicate the validity of a pixel and its location in the frame.

In Simulink, the control signal port is a nonvirtual bus data type called `pixelcontrol`. For details of the bus data type, see “Pixel Control Bus”.

In MATLAB, the control signal argument is a structure. For details of the structure data type, see “Pixel Control Structure”.

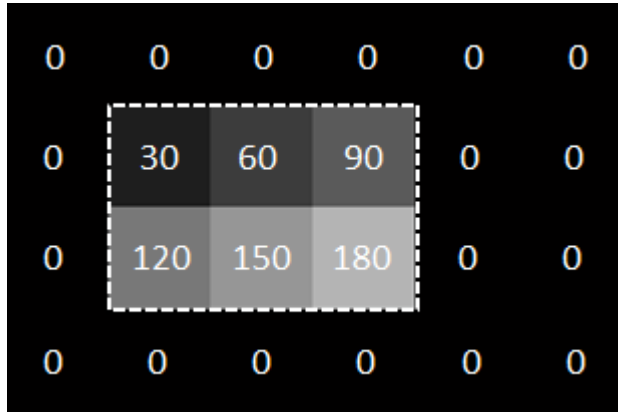
Timing Diagram of Serial Pixel Interface

To illustrate the streaming pixel protocol, this example converts a frame to a sequence of control and data signals. Consider a 2-by-3 pixel image. To model the blanking intervals, configure the serialized image to include inactive pixels in these areas around the active image:

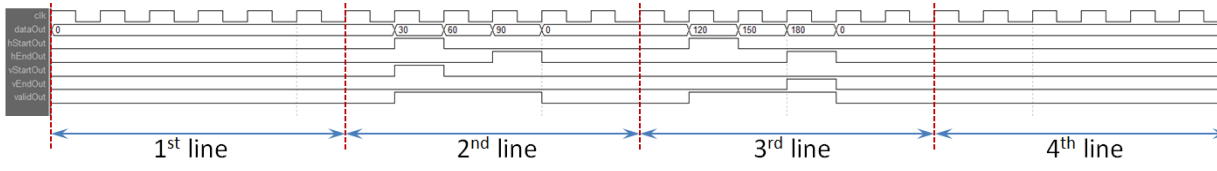
- 1-pixel-wide back porch
- 2-pixel-wide front porch
- 1 line before the first active line
- 1 line after the last active line

You can configure the dimensions of the active and inactive regions with the Frame To Pixels block or `FrameToPixels` object.

In the figure, the active image area is in the dashed rectangle, and the inactive pixels surround it. The pixels are labeled with their grayscale values.



The block or object serializes the image from left to right, one line at a time. The timing diagram shows the control signals and pixel data that correspond to this image. The diagram shows the serial output of the Frame To Pixels block for this frame.



For an example using the Frame to Pixels block to serialize an image, see “Design Video Processing Algorithms for HDL in Simulink”.

For an example using the `FrameToPixels` object to serialize an image, see “Design Video Processing Algorithms for HDL in MATLAB”.

See Also

`visionhdl.FrameToPixels` | `visionhdl.PixelsToFrame` | Frame To Pixels | Pixels To Frame

Pixel Control Bus

Vision HDL Toolbox blocks use a nonvirtual bus data type, `pixelcontrol`, for control signals associated with serial pixel data. The bus contains 5 `boolean` signals indicating the validity of a pixel and its location within a frame. You can easily connect the data and control output of one block to the input of another, because Vision HDL Toolbox blocks use this bus for input and output. To convert an image into a pixel stream and a `pixelcontrol` bus, use the Frame to Pixels block.

Signal	Description	Data Type
`hStart`	`true` for the first pixel in a horizontal line of a frame	`boolean`
`hEnd`	`true` for the last pixel in a horizontal line of a frame	`boolean`
`vStart`	`true` for the first pixel in the first (top) line of a frame	`boolean`
`vEnd`	`true` for the last pixel in the last (bottom) line of a frame	`boolean`
`valid`	`true` for any valid pixel	`boolean`

See Also

Frame To Pixels | `pixelcontrolbus` | Pixels To Frame

More About

- “Streaming Pixel Interface”

Pixel Control Structure

Vision HDL Toolbox System objects use a structure data type for control signals associated with serial pixel data. The structure contains five `logical` signals indicating the validity of a pixel and its location within a frame. You can easily connect the data and control output of a `step` method to the input of another `step` method, because Vision HDL Toolbox objects use this structure for input and output. To convert an image into a pixel stream and control signals, use the `FrameToPixels` object.

Signal	Description	Data Type
`hStart`	`true` for the first pixel in a horizontal line of a frame	`logical`
`hEnd`	`true` for the last pixel in a horizontal line of a frame	`logical`
`vStart`	`true` for the first pixel in the first (top) line of a frame	`logical`
`vEnd`	`true` for the last pixel in the last (bottom) line of a frame	`logical`
`valid`	`true` for any valid pixel	`logical`

See Also

`visionhdl.FrameToPixels` | `visionhdl.PixelsToFrame` | `pixelcontrolsignals` | `pixelcontrolstruct`

More About

- “Streaming Pixel Interface”

2

Video Interfaces and Design Best Practices

Accelerate a MATLAB Design With MATLAB Coder

Vision HDL Toolbox designs in MATLAB must call the `step` method of one or more System objects for every pixel. This serial processing is efficient in hardware, but is slow in simulation. One way to accelerate simulations of these objects is to simulate using generated C code rather than the MATLAB interpreted language.

Code generation accelerates simulation by locking down the sizes and data types of variables inside the function. This process removes the overhead of the interpreted language checking for size and data type in every line of code. You can compile a video processing algorithm and test bench into MEX functions, and use the resulting MEX file to speed up the simulation.

To generate C code, you must have a MATLAB Coder™ license.

See Accelerate a Pixel-Streaming Design using MATLAB Coder.

3

Prototyping

HDL Code Generation from Vision HDL Toolbox

In this section...

What is HDL Code Generation?

You can use MATLAB and Simulink for rapid prototyping of hardware designs. Vision HDL Toolbox blocks and System objects provide support for HDL code generation when used with HDL Coder™. HDL Coder generates target-independent synthesizable Verilog and VHDL code for FPGA programming or ASIC prototyping and design.

HDL Code Generation Support in Vision HDL Toolbox

All blocks and objects in Vision HDL Toolbox, except for Frame To Pixels and Pixels To Frame conversions, are supported for HDL code generation.

Streaming Pixel Interface in HDL

The streaming pixel bus and structure data type used by Vision HDL Toolbox blocks and System objects is flattened into separate signals in HDL. In VHDL it is declared like this:

```
PORT( clk                                 :   IN    std_logic;
      reset                               :   IN    std_logic;
      enb                                 :   IN    std_logic;
      in0                                 :   IN    std_logic_vector(7 DOWNTO 0);  -- u
      in1_hStart                          :   IN    std_logic;
      in1_hEnd                            :   IN    std_logic;
      in1_vStart                          :   IN    std_logic;
      in1_vEnd                            :   IN    std_logic;
      in1_valid                           :   IN    std_logic;
      out0                                :   OUT   std_logic_vector(7 DOWNTO 0);  -- u
      out1_hStart                         :   OUT   std_logic;
      out1_hEnd                           :   OUT   std_logic;
      out1_vStart                         :   OUT   std_logic;
      out1_vEnd                           :   OUT   std_logic;
      out1_valid                          :   OUT   std_logic
```

```
        );
```

In Verilog the interface is declared as follows.

```
input   clk;
input   reset;
input   enb;
input   [7:0] in0;  // uint8
input   in1_hStart;
input   in1_hEnd;
input   in1_vStart;
input   in1_vEnd;
input   in1_valid;
output  [7:0] out0;  // uint8
output  out1_hStart;
output  out1_hEnd;
output  out1_vStart;
output  out1_vEnd;
output  out1_valid;
```

Blocks and System Objects Supporting HDL Code Generation

All blocks and objects in Vision HDL Toolbox, except for Frame To Pixels and Pixels To Frame conversions, are supported for HDL code generation. This page helps you find blocks and objects supported for HDL code generation in other MathWorks® products.

Blocks

You can find libraries of blocks supported for HDL code generation in the Simulink library browser. Find Simulink blocks that support HDL code generation, in the 'HDL Coder' library. You can also type `hdlsllib` at the MATLAB command prompt to open this library.

Create a library of HDL-supported blocks from all products you have installed, by typing `hdllib` at the MATLAB command line. This command requires an HDL Coder license. For more information on this command, see `hdllib` in the HDL Coder documentation.

Refer to the "Supported Blocks" pages in HDL Coder documentation for block implementations, properties, and restrictions for HDL code generation.

System Objects

To find System objects supported for HDL code generation, see Predefined System Objects in the HDL Coder documentation.

Generate HDL Code From Simulink

Introduction

This example generates HDL code from the design described in the Vision HDL Toolbox Getting Started tutorial. See “Design Video Processing Algorithms for HDL in Simulink”. This example generates HDL code from the HDL Algorithm block in the model.

To generate HDL code, you must have an HDL Coder license.

Prepare Model

Run `hdlsetup` to configure the model for HDL code generation. If you started your design using the Vision HDL Toolbox Simulink Model Template, this step is included in the template configuration.

Generate HDL Code

Right-click on the HDL Algorithm block, and select **HDL Code** > **Generate HDL from subsystem** to generate HDL using the default settings. The output of this operation is shown in the MATLAB Command Window.

```
### Generating HDL for 'template_hdlcodegen/HDL Algorithm'.
### Starting HDL check.
### Begin VHDL Code Generation for 'template_hdlcodegen'.
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/FIFOHandlerFSM
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/dataReadFSM as
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/dataWriteFSM a
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/lineSpaceAvera
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/DataMemory/Pus
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/DataMemory/FIF
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/DataMemory/FIF
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/DataMemory/FIF
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/DataMemory/FIF
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/DataMemory/FIF
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/DataMemory/FIF
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/DataMemory/FIF
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/DataMemory/FIF
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/DataMemory/adv
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/DataMemory/isL
```

```
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/DataMemory/unl
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/DataMemory as
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/controlCache a
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/validPipeline
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/horizontalPadd
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/horizontalMux
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer/verticalMux as
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/LineBuffer as hdl_prj\hdl
### Working on template_hdlcodegen/HDL Algorithm/Image Filter/FIR2DKernel as hdl_prj\hd
### Working on template_hdlcodegen/HDL Algorithm/Image Filter as hdl_prj\hdlsrc\templat
### Working on template_hdlcodegen/HDL Algorithm as hdl_prj\hdlsrc\template_hdlcodegen\
### Generating package file hdl_prj\hdlsrc\template_hdlcodegen\HDL_Algorithm_pkg.vhd.
### Creating HDL Code Generation Check Report HDL_Algorithm_report.html
### HDL check for 'template_hdlcodegen' complete with 0 errors, 0 warnings, and 0 messa
### HDL code generation complete.
```

To change code generation options, or to select a target device or synthesis tool, right-click on the HDL Algorithm block, and select **HDL Code > HDL Workflow Advisor**.

Related Examples

- “Generate Code Using the HDL Workflow Advisor”
- “Generate HDL Code Using the Command Line”

Generate HDL Code From MATLAB

Introduction

This example generates HDL code from the design described in the Vision HDL Toolbox Getting Started tutorial. See “Design Video Processing Algorithms for HDL in MATLAB”.

To generate HDL code, you must have an HDL Coder license.

Create an HDL Coder Project

Copy the relevant files to a temporary folder.

```
functionName = 'HDLTargetedDesign';
tbName = 'VisionHDLTB';
vhtExampleDir = fullfile(matlabroot, 'toolbox', 'visionhdl', 'help', 'examples');
workDir = [tempdir 'vht_matlabhdl_ex'];

cd(tempdir);
[~, ~, ~] = rmdir(workDir, 's');
mkdir(workDir);
cd(workDir);

copyfile(fullfile(vhtExampleDir, [functionName,'.m*']), workDir);
copyfile(fullfile(vhtExampleDir, [tbName,'.m*']), workDir);
```

Create a new HDL Coder project. This command opens the HDL Coder app.

```
coder -hdlcoder -new vht_matlabhdl_ex
```

In the HDL Code Generation sidebar, add the function file, `HDLTargetedDesign.m`. and the test bench file, `VisionHDLTB.m`, to the project.

Define the data types for the input and output signals of the function. The control signals are `logical` scalar. The pixel data type in this example is `uint8`. The pixel input is a scalar.

Generate HDL Code

Click the Workflow Advisor button to open the advisor and select options for HDL code generation.

To set code generation options and generate HDL code:

1. Click the 'Code Generation' step to view the HDL code generation options panel.
2. In the Target tab, choose 'Verilog' or 'VHDL' as the 'Language' option.
3. Select the 'Generate HDL' and 'Generate HDL test bench' options.
4. In the 'Coding style' tab, choose 'Include MATLAB source code as comments' and 'Generate report' to generate a code generation report with comments and traceability links.
5. Click the 'Run' button to generate both the Verilog design and test bench with reports.

Examine the log window and click the links to explore the generated code and the reports.

Related Examples

- "Getting Started with MATLAB to HDL Workflow"
- "Generate HDL Code from MATLAB Code Using the Command Line Interface"
- "HDL Code Generation for System Objects"
- Pixel-Streaming Design in MATLAB

HDL Cosimulation

HDL cosimulation links an HDL simulator with MATLAB or Simulink. This communication link enables integrated verification of the HDL implementation against the design. To perform this integration, you need an HDL Verifier™ license. HDL Verifier cosimulation tools enable you to:

- Use MATLAB or Simulink to create test signals and software test benches for HDL code
- Use MATLAB or Simulink to provide a behavioral model for an HDL simulation
- Use MATLAB analysis and visualization capabilities for real-time insight into an HDL implementation
- Use Simulink to translate legacy HDL descriptions into system-level views

More About

- "HDL Cosimulation"

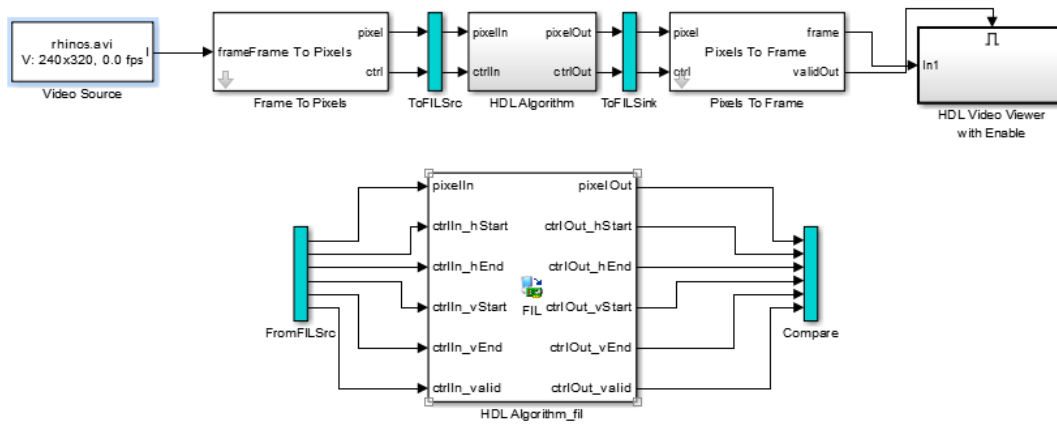
FPGA-in-the-Loop

FPGA-in-the-loop (FIL) enables you to run a Simulink or MATLAB simulation synchronized with an HDL design running on an Altera® or Xilinx® FPGA board. This link between the simulator and the board enables you to verify HDL implementations directly against algorithms in Simulink or MATLAB. You can apply real-world data and test scenarios from Simulink or MATLAB to the HDL design on the FPGA.

Vision HDL Toolbox provides the FIL Frame To Pixels and FIL Pixels To Frame blocks to accelerate communication between Simulink and the FPGA board.

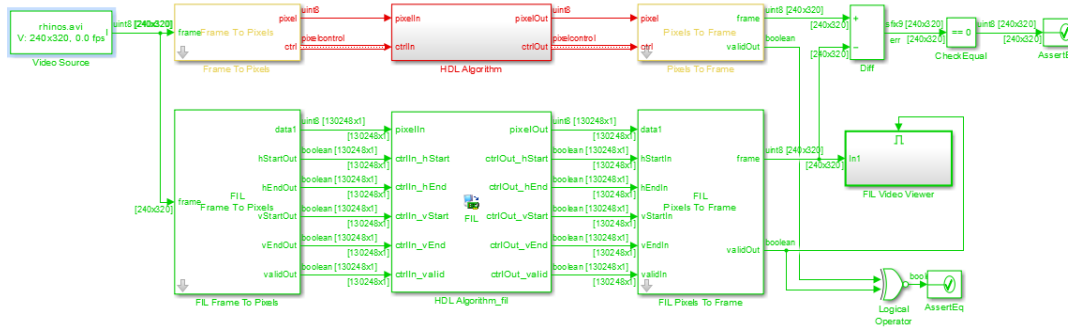
Using FIL blocks

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. The automatic model is shown in the diagram.



The blue ToFILSrc bar branches the pixel-stream input of the HDL Algorithm block to the FromFILSrc bar on the input of the HDL Algorithm_fil block. The second blue bar branches the pixel-stream output of the HDL Algorithm block into the Compare subsystem, where it is compared with the output of the HDL Algorithm_fil block. For image and video processing, this setup can be slow because it sends only a single pixel, and its associated control signals, in each packet to and from the FPGA board.

You can modify the autogenerated model to use the FIL Frame To Pixels and FIL Pixels To Frame blocks to improve the bandwidth of the communication with the FPGA board by sending one frame at a time.



Remove the blue bars, and branch at the full-frame input of the Frame To Pixels block. Insert the FIL Frame To Pixels block before the HDL Algorithm_fil block. Insert the FIL Pixels To Frame block after the HDL Algorithm_fil block. Branch the full-frame output of the Pixels To Frame block for comparison. You can compare the entire frame at once with a Diff block. Use an XOR to compare the `validOut` signals.

Set the **Video format** in the FIL Frame To Pixels and FIL Pixels To Frame blocks to match the Frame To Pixels and Pixels To Frame blocks. You can select `Frame` or `Line` as the vector size. The size of the FIL Frame To Pixels vector output must match the size of theFIL Pixels To Frame vector input. The vector size of the interfaces of the FIL block does not modify the generated HDL code. It only affects the packet size of the communication between the simulator and the FPGA board.

This new model sends an entire frame to the FPGA board in each packet, significantly improving the efficiency of the communication link.

More About

- “FPGA Verification”