

Bringing Imaging to the System Level with PixelStreams

You can develop FPGA-based HDTV and machine vision systems in minutes using Celoxica's suite of ESL tools and the PixelStreams video library.

by Matt Aubury
Vice President, Engineering
Celoxica
matt.aubury@celoxica.com

The demand for high-performance imaging systems continues to grow. In broadcast and display applications, the worldwide introduction of HDTV has driven data rates higher and created a need to enhance legacy standard definition (SD) content for new displays. Simultaneously, rapid changes in display technology include novel 3D displays and new types of emissive flat panels.

Concurrently, machine vision techniques that were once confined to manufacturing and biomedical applications are finding uses in new fields. Automotive applications include lane departure warning, drowsy driver detection, and infrared sensor fusion. Security systems currently in development can automatically track people in closed-circuit TV images and analyze their movements for certain patterns of behavior. In homes, video-enabled robots will serve as entertainment and labor-saving devices.

Cost, processing power, and time to market are critical issues for these new applications. With flexible data paths, high-performance arithmetic, and the ability to reconfigure on the fly, FPGAs are increasingly seen as the preferred solution. However, designers of reconfigurable imaging systems face some big hurdles:

- Getting access to a rich library of reusable IP blocks
- Integrating standard IP blocks with customized IP
- Integrating the imaging part of the system with analysis, control, and networking algorithms running on a CPU such as a PowerPC™ or Xilinx® MicroBlaze™ processor

In this article, I'll introduce PixelStreams from Celoxica, an open framework for video processing that has applicability in both broadcast and machine vision applications. PixelStreams leverages the expressive power of the Celoxica DK Design Suite to develop efficient and high-performance FPGA-based solutions in a fraction of the time of traditional methods.

Filters and Streams

A PixelStreams application is built from a network of filters connected together by streams. Filters can generate streams (for example, a sync generator), transform streams (for example, a colorspace converter), or absorb streams (for example, video output).

Streams comprise flow control, data transport, and high-level type information. The data component in turn contains:

- An active flag (indicating that this pixel is in the current region-of-interest)
- Optional pixel data (in 1-bit, 8-bit, or signed 16-bit monochrome, 8-bit YCbCr, or RGB color)
- An optional (x, y) coordinate
- Optional video sync pulses (horizontal and vertical sync, blanking, and field information for interlaced formats)

Combining all of these components into a single entity gives you great flexibility.

Filters can perform purely geometric transforms by modifying only the coordinates, or create image overlays just by modifying the pixel data.

Filters use the additional type information that a stream provides in two ways: to ensure that they are compatible with the stream they are given (for example, a sync generator cannot be attached directly to a video output because it does not contain any pixel data), and to automatically parameterize themselves. Filters are polymorphic; a single PxsConvert() filter handles colorspace conversion between each of the pixel data formats (which includes 20 different operations).

Flow Control

Flow control is handled by a downstream "valid" flag (indicating the validity of the data component on a given clock cycle) and an upstream "halt" signal. This combination makes it easy to assemble multi-rate designs, with pixel rates varying (dynamically if necessary) from zero up to the clock rate. Simple filters are still easy to design. For example, a filter that modifies only the pixel components would use a utility function to copy the remaining components (valid, active, coordinates, and sync pulses) from its inputs to outputs while passing the halt signal upstream. More complex filters that need to block typically require buffering at their inputs, which is easily handled by using the generic PxsFIFO() filter.

Sophisticated designs will require filter networks in multiple clock domains. These are easy to implement using the DK Design Suite, which builds efficient and reliable channels between clock domains and sends and receives filters that convert streams to channel communications. The first domain simply declares the channel (in this case, by building a short FIFO to maximize throughput) and uses the PxsSend() filter to transmit the stream:

```
// domain 1
chan X with { fifolength = 16 };
...
PxsSend (&S0, &X);
```

The second domain imports the channel using extern and receives the stream from it with a PxsReceive() filter:

```
// domain 2
extern chan X;
...
PxsReceive (&S1, &X);
```

Custom Filters

PixelStreams comes "batteries included," with more than 140 filters, many of which can perform multiple functions. A sample of this range is shown in Table 1. Every PixelStreams filter is provided with complete source code. Creating custom filters is often just a matter of modifying one of the standard filters. New filters are accessible from either the graphical editor or from code.

- Image analysis and connected component ("blob") labeling
- Image arithmetic and blending
- Clipping and region-of-interest
- Colorspace conversion and dithering
- Coordinate transforms, scaling, and warping
- Flow control, synchronization, multiplexing, FIFOs, and cross-clock domain transport
- Convolutions, edge detection, and non-linear filtering
- Frame buffering and de-interlacing
- Color look-up-tables (LUTs)
- Grayscale morphology
- Noise generators
- Video overlays, text and cursor generators, line plotter
- Sync generators for VGA, TV, HDTV
- Video I/O

Table 1 – An overview of the provided source-level IP

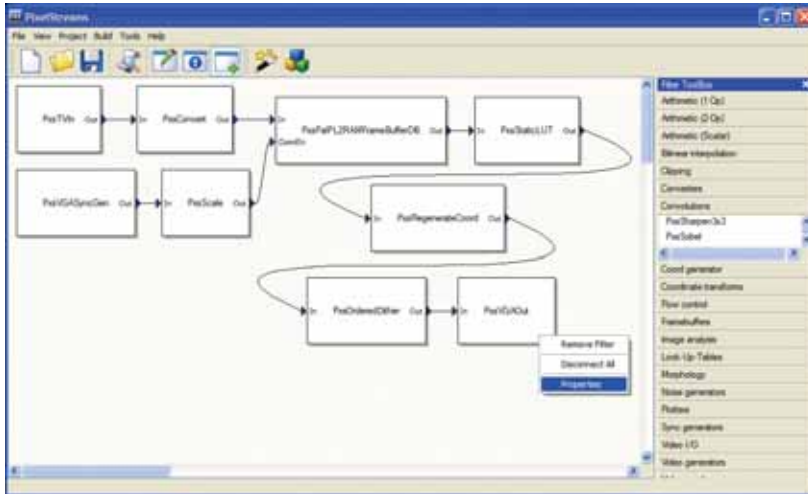


Figure 1 – Example application built using the PixelStreams graphical editor

```
#define ClockRate 65000000
#include "pxs.hch"

void main (void)
{
    /* Streams */
    PXS_I_S (Stream0, PXS_RGB_U8);
    PXS_I_S (Stream1, PXS_YCbCr_U8);
    PXS_PV_S (Stream2, PXS_EMPTY);
    PXS_PV_A (Stream3, PXS_EMPTY);
    PXS_PV_A (Stream4, PXS_RGB_U8);
    PXS_PV_S (Stream6, PXS_RGB_U8);
    PXS_PV_A (Stream7, PXS_RGB_U8);
    PXS_PV_S (Stream8, PXS_RGB_U8);

    /* Filters */
    par
    {
        PxsTVIn (&Stream1, 0, 0, ClockRate);
        PxsConvert (&Stream1, &Stream0);
        PxsPalPL2RAMFrameBufferDB (&Stream0, &Stream3, &Stream4, Width, PXS_BOB,
                                   PalPL2RAMCT(0), PalPL2RAMCT(1), ClockRate);
        PxsVGASyncGen (&Stream2, Mode);
        PxsScale (&Stream2, &Stream3, 704, 576, 1024, 768);
        PxsStaticLUT (&Stream4, &Stream7, PxsLUT8Square);
        PxsOrderedDither (&Stream8, &Stream6, 5);
        PxsVGAOut (&Stream6, 0, 0, ClockRate);
        PxsRegenerateCoord (&Stream7, &Stream8);
    }
}
```

Figure 2 – Generated source code for example application

Concept to FPGA Hardware in 10 Minutes

Let's take a simple HDTV application as an example. We have an SD video stream that we want to de-interlace, upscale, apply gamma correction to, then dither and output to an LCD.

We can assemble the blocks for this very quickly using the PixelStreams graphical editor. The TV (SD) input undergoes color-space conversion from YCbCr to RGB and is stored in a frame buffer that implements simple bob de-interlacing. Pixels are fetched from the frame buffer, with the upscaling achieved by modifying the lookup coordinates. The output stream is gamma-corrected using a color LUT, dithered, and output.

Assembling this network takes only a few minutes. The editor uses heuristics to set the stream parameters automatically. The final design is shown in Figure 1.

One click generates code for this network, with a corresponding project and workspace, and launches the DK Design Suite. The generated code is shown in Figure 2. For this simple application each filter is a single process running in a parallel "par" block. More sophisticated designs will have C-based sequential code running in parallel to the filters.

One more click starts the build process for our target platform (in this case, Celoxica's Virtex™-4 based RC340 imaging board). The build process automatically runs ISE™ software place and route and generates a bit file, taking about five minutes. One last click and the design is downloaded to the board over USB, as shown in Figure 3.

As designs become more complex, it becomes easier to code them directly rather than use the graphical editor. This enables a higher level of design abstraction, with the ability to construct reusable hierarchies of filters and add application control logic.

Simulation

Simulation is handled by the DK Design Suite's built-in cycle-based simulator and utilizes its virtual platform technology to show video input and output directly on screen. Celoxica's co-simulation manager can simulate designs incorporating CPUs and external RTL components.

Future versions of PixelStreams will extend both its flexibility as a framework and the richness of the base library of IP.

Synthesis

PixelStreams capitalizes on the strengths of the DK Design Suite's built-in synthesis to achieve high-quality results on Xilinx FPGAs. For example:

- When building long pipelines, it is common that certain components of streams will not be modified. These will be efficiently packed into SRL16 components.
- When doing convolutions or coordinate transforms, MULT18 or DSP48 components will be automatically targeted (or if the coefficients are constant, efficient constant multipliers are built).
- Line buffers and FIFOs will use either distributed or pipelined block RAM resources, which are automatically tiled.
- The data components of a stream are always registered, to maximize the timing isolation between adjacent filters.
- DK's register retiming can move logic between the pipeline stages of a filter to maximize clock rate (or minimize area).

In addition, DK can generate VHDL or Verilog for use with third-party synthesis tools such as XST or Synplicity and simulation tools such as Aldec's Active HDL or Mentor Graphics's ModelSim.

System Integration

A complete imaging system typically comprises:

- A PixelStreams filter network, with additional C-based control logic
- A MicroBlaze or PowerPC processor managing control, user interface, or high-level image interpretation functions
- VHDL or Verilog modules (possibly as the top level)

To deal with CPU integration, PixelStreams provides a simple "PxsBus" that can be readily bridged to CPU buses such as AMBA or OPB or controlled remotely (over PCI or USB). This is purely a control bus, allowing filters to be controlled by a CPU (for adding menus or changing filter coefficients) or to provide real-time data back from

a filter to the controlling application (such as the result of blob analysis).

To support integration with RTL flows, PixelStreams offers a PxsExport() filter that packages a filter network into a module that can be instantiated from VHDL or Verilog. Alternatively, an RTL module can be instantiated within a PixelStreams top level using PxsImport(). Used together, pre-synthesized filter networks can be rapidly instantiated and reduce synthesis time.

Conclusion

Combining the two main elements of ESL tools for Xilinx FPGAs – C- and model-based design – PixelStreams offers a uniquely powerful framework for implementing a variety of imaging applications. The provision of a wide range of standard filters combined with features for integration into RTL flows and hardware/software co-design makes it easy to add imaging features to your system-level designs.

Future versions of PixelStreams will extend both its flexibility as a framework and the richness of the base library of IP. We plan to add additional pixel formats (such as Porter-Duff based RGBA) and color depths and increase the available performance by introducing streams that transfer multiple pixels per clock cycle. We also intend to introduce the ability to easily transmit streams over other transports, such as USB, Ethernet, and high-speed serial links, add more filters for machine vision features such as corner detection and tracking, and offer additional features for dynamically generating user interfaces.


You can use PixelStreams to target any prototyping or production board identified for a project with the addition of simple I/O filters. It can immediately target Celoxica's range of RC series platforms and is fully compatible with the ESL Starter Kit for Xilinx FPGAs. For more information, visit www.celoxica.com/xilinx, www.celoxica.com/pxs, and www.celoxica.com/rc340. 



Figure 3 – Application running in hardware