PixelStreams and Tetris

David Edell

Spring 2006 – Dr. Vickery

I. Introduction

The PixelStreams Library is a powerful tool in the Celoxica toolkit. It allows for the streamlining and automation of a wide variety of video related functions. Each function is implemented as a filter. A series of filters can be linked together to create (or rather using) a stream, with the input of each stream connected to the output of the previous one. A variety of filters are predefined (and described in the PixelStreams documentation), however you may also define your own custom filter for later use.

PixelStreams are used in place of direct manipulation of the PalVideo and related interfaces. Filters can be used to process video input, such as from a camera or computer. That video input can be combined with other video signals and sent output to an available video output device. Filters are also provided for creating new video streams, applying transformations, overlaying text or other objects, merging or splitting video signals, as well as for implementing a frame buffer.

A GUI utility is provided by Celoxica as a method of starting a new application using streams. The utility allows one to graphically place and connect individual filters to create streams. The utility is useful for quick testing of concepts, or for generating code to use as a basis for further projects. In order for the generated project to work however, the streams must be created and placed in the proper order and with the correct types. After learning the basics of the PixelStreams library however, it is best to generate the code by hand.

II. PixelStreams Basic Usage

The PixelStreams library requires two additional libraries to be added under the linker. *Pxs.hcl* is required for both EDIF and Debug. Note that while the PixelStreams library does support simulations, not all filters do. The library *pxs_sim.hcl* is required for simulation, and *pxs_xxxxx.hcl* for debug, where *xxxx* is the platform name (ie: rc200e or rc300). A series of macro constants must also be declared at the start of a PixelStreams programs to define the video mode, clock rate, and display size.

```
/** Mode macros **/
macro expr ClockRate = PAL_ACTUAL_CLOCK_RATE;
macro expr Mode      = SyncGen2GetOptimalModeCT   (ClockRate);
macro expr Width     = SyncGen2GetHActivePixelsCT (Mode);
macro expr Height    = SyncGen2GetVActiveLinesCT  (Mode);
```

There are two main components to a PixelStreams project; the streams and the filters. Streams should be declared at the start of the main method using a macro call of the following format:

PXS_PV_S (VGASync,   PXS_EMPTY);

The first parameter is simply the name of the stream. This becomes declared as a variable which will be referenced in the filters later. These variables are later filled, and passed along, by the filters. The second parameter is the stream type. *PXS_EMPTY*, as the name suggests, is an empty stream. In most cases, the first stream in the file will be empty. Filters may accept or return filters of different parameters, and must be given a reference(s) to a stream of the correct type. *PXS_RGB_U8* is the standard type used for the devices video display. Other types include *PXS_YCbCr_U8* used by the S-Video and webcam inputs.

References to these stream objects are passed to filters. Filters have a general syntax of

***PxsFilterName(&In,&Out,Params)*.**

Some filters may omit, or have extra, inputs or outputs. Parameters can be any variable, constant, or other requirements of individual filters. In refers to the input stream, and output to the output stream. Sync generators, as well as inputs, create (or fill) a new stream, and therefore do not have an *In*. Generally, the *Out* of one filter will be connected to the *In* of the next. This is done by passing the same Stream variable at each connecting point. Stream variables cannot be reused or connected to more than one In and Out (at most one of each) at a time.

Filters are each specified with certain allowed stream types for any input and output streams. SyncGen streams generated a clock synchronization pulse that synchronizes later streams to the specified video source. For example, the following initializes a stream for output to the LCD display:

PxsVGASyncGen   (&VGASync, Mode);

The last filter in a stream series outputs the result to the display. For the LCD, it is:

PxsVGAOut   (&Output, 0, 0, ClockRate);

The majority of filters require their input to have a non-empty stream type. SyncGen initializes a stream, but does not change its type. Other filters, such as PxsCheckerboard and others in the pxs_video_gen library can generate background images and initialize a streams type.

The *PxsSplit* filter can be used to temporarily split (clone) a filter into two parts. Those parts can be directed to separate output devices (if available), or remerged later using filters such as *PxsAverage*, which will average the pixel values (at each coordinate) of the two streams together. This allows for the creation of some interesting effects and patterns, as can be seen in the PixelStreams examples.

Translating a video stream from an input source to an output source requires the usage of several filters in series. Input and Output streams must each be initialized separately (as above). If the two sources are not of the same Stream type, then the *PxsConvert* filter may be applied to change the type of the input stream to that of the output stream. To do this, the *In* and *Out* stream variables must be of different, but compatible, types. Conversion filters are the only cases where the *In* and *Out* filters may be of different types.

Video Input and Video Output streams rarely share the same synchronization type, and therefore cannot be connected directly. A *PxsFIFO* filter can be used in this case to create a first-in-first-out queue that buffers the video input signal. The output could then be passed to a FrameBuffer to ensure synchronization between the input and output signals. A FrameBuffer accepts a video input stream to buffer, a second input stream containing the output synchronization type (in other words, a filter connected to a VGASync), and outputs the buffers contents to the *Out* stream. Framebuffers are available in the PixelStreams library, including those that use a single PalPL1RAM. Framebuffers are also available that use multiple PalPL1RAMs, allowing output to be toggled between buffers while other computations are performed on the signal.

Any number of filters may be inserted between the initialized input and final output streams. These include a PixelStreams text console, mouse pointers, and other objects that can be overlaid on the screen. Other filters can be used to resize or rotate the screens contents, or to otherwise manipulate the picture, including blue effects or modifying the colors.

The PixelStreams library provides additional macro functions that do not affect the contents of a stream. These functions are defined in *pxs_utilities* library in the PixelSreams Sources directory (*C:\Program Files\Celoxica\PDK\Hardware\Source \PixelStreams*). The function *PxsAwaitVSync (&VGASync)* can be used anywhere in your program to delay calculation until the next screen synchronization (refresh), given a reference to a valid and initialized stream. In addition, some library functions, such as the PxsConsole, provide macros for writing data to the filter. See the documentation of the filters for more details.

III. PixelStreams Custom Filters

The PixelStreams library provides an easy set of tools for creating custom filters. These filters are defined as macro functions, and can be placed in your main source code or within custom libraries. The filters provide a layer of abstraction from the mechanics of the video interface. Within the filter, one needs to be concerned only with the actions to be performed by this specific filter, and on only one pixel at a time.

Filters follow a simple format, and except for their declaration, are written identically to those provided. The provided filters (*C:\Program Files\Celoxica\PDK \Hardware\Source\PixelStreams*) can serve as a good starting point when creating custom filters. The following is a sample custom filter declaration, based on the provided PxsRectangle filter in the *pxs_overlay* library:

```
macro proc PxsCustom_Rect( In, Out, X0, Y0, X1, Y1, Color1 )
{
   PXS_DECLARE_BW (Out);
   PXS_EXPECT_COORD (In);
   PXS_OVERLAY (In, Out);
   while (1)
   {
      par
      {
         PxsCopyVCSH (In, Out);
         if ((In->Coord.X >= X0 && In->Coord.X <= X1) &&
            (In->Coord.Y >= Y0 && In->Coord.Y <= Y1))
         {
            par
            {
               Out->Pixel = (Color1 ? PxsWhite : PxsBlack);
               Out->Active = !In->Sync.Blank;
            }
         }
         else
         {
            par
            {
               Out->Pixel = In->Pixel;
               Out->Active = In->Active;
```

```
                }
              }
            }
          }
        }
```

In and *Out* are the input and output streams, respectively, as described previously. *In->Coord* gives the current X and Y co-ordinates of the active pixel position that you can write to. The PixelStreams library handles the details of the video display, ensuring that the specified coordinates correspond to the pixel that you are writing to.

All actions performed in a single stage of a filter must be atomic – they must take exactly one clock cycle. Additionally, all stages of the filter are performed in parallel, forming a pipeline. In the above example there is only one stage, the initial one. We advance from the old stage (*In)* to the final one (*Out)* through a call to *PxsCopyVCSH(In,Out)*. Additional stages may also be created, using a syntax similar to that of creating the filters themselves. Stages can be declared by:

PXS_GENERIC(StageX);

Where *StageX* is the name of the stage variable. A filter can have any number of Stages, each maintaining its own Coord, Pixel, and Active values. Each Stage corresponds to a step in the display pipeline. All of the filters in a stream perform their calculations along this pipeline, building on the values of its predecessors.

For example, to use a two stage pipeline in a custom filter, simply declare a Stage0 generic stream. To progress between stages, calls of *PxsCopyVCSH(In,Stage0)*, and *PxsCopyVCSH(Stage0,Out)* are used. Calculations at each stage should refer only to constants or variables declared as parameters to the stream, local variables set in the previous stage, and the values stored in the Stage variable itself.

The value of *Stage->Pixel* contains the color of the current location. If a filter does not perform any modifications to the current location, it should set *In->Pixel* equal to *Out->Pixel* between stages. *PxsWhite* and *PxsBlack* are constants defined in the library.

Parameters may be passed to a custom filter like any other macro function. Certain functions, as in the example above, are used to validate that parameters are of the correct type at compile-time. No documentation appears to exist on these parameter and stream type declaration functions; however their meanings are usually self-explanatory when viewed in context of the provided filters. Those functions that do not modify the behavior of the stream are primarily for error checking only, and can usually be safely omitted.