Summary

By temporarily storing and providing access to video data in frame buffers and line buffers, video algorithms can utilize scene coherence in digital signal processing algorithms. This application note provides technical details surrounding temporary storage implemented in Xilinx FPGAs and shows how to effectively process video using the MicroBlaze™ and Multimedia development board.

Introduction

There are many graphics and video systems using Application Specific Standard Product (ASSP) devices to implement a line buffer function. There are also special memory devices specifically available for buffering video frame data. This application note helps designers integrate these and many other video functions inside a single Virtex™ or Spartan™-II device. It also describes how to replace expensive special-purpose memories used for frame storage, by interfacing to standard inexpensive memory devices.

There are many ASSP video line buffer and frame buffer memory devices. Here are just a few:

- Logic Devices LF3304 4K x 12 Dual Line Buffer/FIFO
- Logic Devices LF9501 1280 x 10-bit Programmable Line Buffer
- Logic Devices LF9502 2048 x 10-bit Programmable Line Buffer
- Logic Devices LF3312 is a 12.5 Mb Frame Buffer memory and FIFO
- NEC µPD485506 5K x 16 Line Buffer

The line buffers have about the same density as a Virtex-II block RAM. To architect a video or graphics system in a Xilinx FPGA, the following should be considered:

1. How do real objects, represented as video data, exhibit coherence on several levels? What does this mean to video, graphics, and image processing?
2. How is video data accessed for display and implications on processing?
3. What are the data rates and sizes for various video components?
4. What is the best way to implement the necessary designs in a Xilinx FPGA?

This application note touches on all four of these topics. Two versions of line buffers, one using block RAM and one using SRL16s are discussed as well as two versions of a frame buffer interface, namely ZBT RAM and DDR SDRAM. The reference designs will use both line buffers interfaced to a ZBT RAM controller. The DDR SDRAM controller is available in a separate application note (XAPP200).
"Live" 3D Objects

2D Video Representations Exhibit Coherence

Coherence is explained in the book "Computer Graphics Principles and Practice"\(^1\) quoting Sutherland, Sproull, and Schumacher\(^2\):

"Coherence – the degree to which parts of an environment or its projection exhibit local similarities. Environments typically contain objects whose properties vary smoothly from one part to another. In fact, it is the less frequent discontinuities in properties (such as depth, color, and texture) and the effects that they produce in pictures, that let us distinguish between objects."

When considering natural physical properties and how those properties map into screen space representations, video and graphics designers find coherence in the horizontal, vertical, and temporal domains very interesting.

A fourth domain pointed out by Charles Poynton in "Digital Video" is termed spatial. The spatial domain concept is a screen area algorithm with no ability to be separated into its horizontal and vertical parts. Some computer graphics anti-aliasing algorithms, for example, fall into this category. There are many other domains not covered in this application note. Figure 1 shows a graphic representation of these four domains.

---

Video Data Considerations

Displaying Video Data

Video is physically drawn on a CRT by sweeping an electron beam across the surface from left to right. As it sweeps the surface, the beam energizes specific color triads one video line at a time, defining small distinctly colored picture elements known as PELs or pixels. When the end of a line or far right-hand side of the screen is reached, the beam is turned off or blanked and returned to the starting point, only moved down by a single line width. After the entire CRT is swept in this fashion, left to right and top to bottom, the beam is turned off and not only returned to the left side of the screen, but also to the top, completing the single frame or field. Drawing many frames very fast allows the human visual system to integrate the individual pixels into pictures and integrate the pictures providing scene motion (a temporal effect).
The way video is drawn on a CRT, since pixels can exhibit coherence in various domains, leads to the control and data circuits. In other words, data is processed several localized pixels at a time, several localized scan lines at a time, or several localized frames at a time. The algorithms must have access to several adjacent pixels, scan lines, or frames. Memories must also accommodate these requirements.

**Video Data Rates and Data Sizes**

The size of video data elements and data rates for broadcast video and image processing are shown in Table 1. Video components of interest are the length of the video line in terms of pixels, the number of video lines in the vertical screen dimension, and the number of frames drawn per second.

Notice the bit-serial rates in the last column of data. These rates are used in Serial Data Interface (SDI) video transfer standards. The following application notes address SDI:

- **XAPP625**: Serial Digital Interface (SDI) Video Decoder Flywheel
- **XAPP288**: Serial Digital Interface (SDI) Video Decoder
- **XAPP298**: Serial Digital Interface (SDI) Video Encoder
- **XAPP299**: Serial Digital Interface (SDI) Ancillary Data and EDH Processors
- **XAPP247**: Serial Digital Interface (SDI) Video Physical Layer Implementation

In addition to the numbers listed in Table 1, it is important to know, based on the algorithms, how many pixels, lines, or frames an algorithm needs to access. For example, a 422 to 444 conversion algorithm can access 24 consecutive pixels to compute a resulting pixel. De-interlacing lines can access four separate lines to compute a resulting line.

Some algorithms need to access a frame more than once in the allotted frame time. In many cases, more than color is stored in a pixel requiring even higher bandwidth.

---

**Table 1: Bandwidth and Data Size Calculations for Broadcast Digital Video (Y’CrCb 4:2:2)**

<table>
<thead>
<tr>
<th>Video Format</th>
<th>Active Pixels</th>
<th>Total Pixels</th>
<th>Pixels per Frame</th>
<th>Frames per Second</th>
<th>Pixels per Second</th>
<th>Serial Bit Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTSC</td>
<td>720 x 485</td>
<td>858 x 525</td>
<td>450,450</td>
<td>30</td>
<td>13.5 M</td>
<td>270 Mb/s</td>
</tr>
<tr>
<td>PAL</td>
<td>720 x 576</td>
<td>864 x 625</td>
<td>540,000</td>
<td>25</td>
<td>13.5 M</td>
<td>270 Mb/s</td>
</tr>
<tr>
<td>HDTVi</td>
<td>1920 x 1035</td>
<td>2200 x 1125</td>
<td>2.475 M</td>
<td>30</td>
<td>74.25 M</td>
<td>1.485 Gb/s</td>
</tr>
</tbody>
</table>
**Necessary Video Storage Components**

**Registers**

Algorithms using localized groups of pixels along a given line are simple, using coherence in the horizontal direction. XAPP294: Video Digital Component Conversion 4:2:2 to 4:4:4, describes the formation of missing Cr or Cb components, based on the filter contributions from adjacent pixels. One can easily hold 2, 4, 16, or more pixels in flip-flops and not tax today’s FPGA resources. In fact, the largest filter mentioned in the application note requires 24 pixels or 240 flip-flops, assuming 10-bit pixels. Equation 3:

\[
Cb[i] = (-4 \cdot (Cb[i-23] + Cb[i+23]) + 6 \cdot (Cb[i-21] + Cb[i+21]) - 12 \cdot (Cb[i-19] + Cb[i+19]) + 20 \cdot (Cb[i-17] + Cb[i+17]) - 32 \cdot (Cb[i-15] + Cb[i+15]) + 48 \cdot (Cb[i-13] + Cb[i+13]) - 70 \cdot (Cb[i-11] + Cb[i+11]) + 104 \cdot (Cb[i-9] + Cb[i+9]) - 152 \cdot (Cb[i-7] + Cb[i+7]) + 236 \cdot (Cb[i-5] + Cb[i+5]) - 420 \cdot (Cb[i-3] + Cb[i+3]) + 1300 \cdot (Cb[i-1] + Cb[i+1]))/2048;
\]

Figure 3 shows a block diagram implementing the above set of equations. As pixels march through the pipeline, the “phantom pixel” is calculated based on the nearest neighbors along the scan line, thus taking advantage of horizontal coherence. In other words, any given pixel will typically be a similar color to the adjacent pixels in the scan line, so by applying a FIR filter function to those pixels, a “phantom” pixel value can be calculated.

![Figure 3: 422 to 444 FIR Filter Uses Horizontal Scan-Line Coherency](image-url)
Video Scene Coherence, Frame Buffers, and Line Buffers

**Line Buffers**

The amount of storage increases for vertical coherence. Figure 4 shows that in order to look at a vertical stripe of four pixels, thereby exploiting vertical coherence, four lines need to be stored. Conceptually, the line buffer is a synchronous array of registers 8 bits or 10 bits wide (pixel width) and 720 deep (active line length).

![Figure 4: Line Buffers Allow Algorithms to See Vertical Stripe Pixels.](image)

This is easily done in Virtex or Spartan-II families by using either block RAM or LUTs configured as SRL16s. For comparison, Table 2 shows the commercial ASSP device densities versus Virtex-II block RAM or SRL16 implementations.

**Table 2: Memory Solutions for Video Line Buffers**

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>ASSP Memory Types</th>
<th>Organization</th>
<th>Virtex-II Block RAM Utilization</th>
<th>Virtex or Spartan-II SRL16s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Devices LF3304</td>
<td>Video Line Buffer 48K bits</td>
<td>4K x 12</td>
<td>2.7</td>
<td>3000 LUTs</td>
</tr>
<tr>
<td>NEC485506</td>
<td>Video Line Buffer 80K bits</td>
<td>5K x 16</td>
<td>4.4</td>
<td>5000 LUTs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10K x 8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logic Devices LF9501</td>
<td>Video Line Buffer 15K bits</td>
<td>1290 x 12</td>
<td>0.8</td>
<td>938 LUTs</td>
</tr>
<tr>
<td>Logic Devices LF9502</td>
<td>Video Line Buffer 24 K bits</td>
<td>2048 x 12</td>
<td>1.3</td>
<td>1500 LUTs</td>
</tr>
<tr>
<td>NTSC Line 720 pixels Y'CrCb</td>
<td>Video Line Buffer 11520 bits</td>
<td>1440 x 8</td>
<td>1 each 2K x 9</td>
<td>720 LUTs</td>
</tr>
</tbody>
</table>

In the Virtex-II architecture, a standard definition line requires only one block RAM running at a rate of 27 MHz. The two line de-interlace algorithm is an example. You can read the details of how the development board does de-interlace in XAPP285: Video Scan Line De-interlacing. Figure 5 shows the most straightforward implementation of the two line de-interlace. A simpler version, using only a one-line buffer, is shown in Figure 6. This implementation produces two lines at once requiring the output to be twice the data rate as the input. The design adds two small FIFOs that accept the lines at the normal line rate. The output of each FIFO can be run at twice the data rate, filling the ZBT frame buffer appropriately. The most efficient implementation for the small FIFOs is from LUTs used as dual-port memory, a feature found only in Xilinx FPGAs.
Figure 5: Scan Line De-Interlacing Using Two-Line Averaging

Figure 6: Alternate Implementation of Scan Line De-Interlacing Using Two-Line Averaging and One-Line Buffer

Figure 7 and Figure 8 show block diagrams for the Line Buffer using block RAM and SRL16s respectively. Notice that the Line Buffer in SRL16s requires no addressing or special control.
Frame Buffers

These examples show that a basic limitation to leveraging a scene’s coherence is how far the data exhibiting the coherence is displaced in time. For adjacent pixels that are not displaced very far, the process is easy, requiring only a few flip-flops. For two pixels at the same location in adjacent scan lines, the problem is more difficult, requiring an entire line of storage. For two pixels at the same location in two different frames, still more storage is needed. This is one use of the frame buffer. Of course, just storing the pixels waiting their time to be displayed is a minimum requirement.

Some spatial algorithms can be separated into their horizontal and vertical components. Compression is an example of an algorithm that can usually be separated. Compression algorithms look for spatial and temporal coherence to reduce the number of bits communicated or stored. XAPP610 describes a one-dimensional DCT/IDCT. This algorithm can be run in multiple dimensions, independent of each other, to reduce an image into mostly zeros by removing the high frequency spatial components.

On the other hand, anti-aliasing, is an algorithm in computer graphics leveraging spatial coherence that requires a block of XY data and, therefore, cannot be easily separated. Aliasing is a visual artifact of raster systems that arises from the sampling error introduced by any digital system sampling a continuous function. If graphics and live video are mixed, this issue will need to be addressed in the development board.

Figure 9 illustrates smoothing the “jaggies” with an anti-aliasing algorithm. The polygon (black) and background color (white) are augmented by a third color, a combination of the two (gray) to the polygons edge. The human visual system integrates the three colors giving the effect of a "smoother" edge. Many of the anti-aliasing algorithms require an "area" of pixels.
Hidden surface removal is another algorithm requiring multiple reads and writes to the same pixel and, therefore, very high bandwidth in a frame buffer. Hidden surface removal becomes a problem when an image is artificially produced. In real life, of course an object that is closer and is opaque occludes an object that is farther away from a given eye point. This is not the case in 3D graphics mathematics. And the problem grows when mixing real life images and artificial computer-generated images. Figure 10 shows how artificial objects might “occlude” each other and how many “pixel accesses” it would take to resolve the conflicts.

### Multi-Frame Algorithms

#### Memory Density and Bandwidth Requirements

Hidden surface removal and other algorithms leveraging temporal or spatial coherence require very high memory bandwidth. It is not uncommon to have bandwidth requirements that are 10 times the number of pixels in the frame times the frame rate.

Table 3 lists the three most widely used memories for frame buffer design. The density is shown as the number of high definition frames (2.16M pixels x 20 bits = 43 Mb) that can be stored. The bandwidth is expressed in pixel touches where a pixel touch is defined as accessing every pixel in the frame once per frame time (43 Mb x 30 fields/s = 1.3 Gb/s).
The DDR SDRAM is the most optimum choice for a frame buffer in terms of cost, density, and bandwidth. For an algorithm that needs to access every pixel in the frame buffer approximately nine times per frame (i.e., nine pixel touches), one DDR SDRAM, two ZBT RAMs, or one QDR SRAM are needed. For an algorithm that needs to hold about nine frames of data, 43 QDR SRAMs, 24 ZBT RAMs, and six DDR SDRAMs are needed.

Frame Buffer Addressing

Frame buffer addressing is very dependent on what types of data are stored as well as how algorithms need the data presented. A very common requirement is to convert pixels in different positions along a line and at different line counts into memory addresses. One way to think about this is to compose the memory address as:

\[ \text{Pixel Count} \times \text{Line Count} = \text{Memory Address} \]

It should also be mentioned that external memories can be operated in more efficient block transfer modes by taking the data in and out of video algorithms using a Virtex-II block RAM as an intermediate FIFO. This allows the external memories to run at optimum device speed (Figure 11).

---

### Table 3: Memory Solutions For Video Frame Buffers

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Memory Types</th>
<th>Organization</th>
<th>Density in HD Frames and Mbs</th>
<th>Bandwidth in HD Pixel Touches and bits/second</th>
<th>Figure of Merit Cost/BW x Density (low is good)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Micron</td>
<td>167 MHz QDR, DDR, SRAM</td>
<td>512K x 18 (2 ports)</td>
<td>0.21 HD Frames (9.2 Mb)</td>
<td>9.2 Pixel Touches (12 Gb/s)</td>
<td>$30/(0.21 x 9.2) = 15.53</td>
</tr>
<tr>
<td>IDT ZBT</td>
<td>167 MHz, not DDR</td>
<td>512 x 36</td>
<td>0.37 HD Frames (16 Mb)</td>
<td>4.6 Pixel Touches (6 Gb/s)</td>
<td>$60/(0.37 x 4.6) = 35</td>
</tr>
<tr>
<td>Micron</td>
<td>64 Mb, 200 MHz DDR SDRAM</td>
<td>2M x 32</td>
<td>1.49 HD Frames (64 Mb)</td>
<td>9.8 Pixel Touches (12.8 Gb/s)</td>
<td>$12/(1.49 x 9.8) = 0.82</td>
</tr>
</tbody>
</table>

**Notes:**
1. ZBT has fast read-modify-write.
2. ZBT has simple interface.
3. Current pricing through distribution.
4. Video wants x30-bit-wide or x24-bit-wide minimum.

---

The DDR SDRAM is the most optimum choice for a frame buffer in terms of cost, density, and bandwidth. For an algorithm that needs to access every pixel in the frame buffer approximately nine times per frame (i.e., nine pixel touches), one DDR SDRAM, two ZBT RAMs, or one QDR SRAM are needed. For an algorithm that needs to hold about nine frames of data, 43 QDR SRAMs, 24 ZBT RAMs, and six DDR SDRAMs are needed.
Features Used

The high-speed block RAMs and SRL16s in Virtex and Spartan-II families form excellent line buffers as used in the previous examples. As seen in the drawings, 10-bit wide data for each video component Y’CrCb is used in the development board designs. A line of video is 858 pixels for NTSC or 864 pixels for PAL (including blanked pixels). Control is simplified if the blanking pixels are just handled like visible pixels. A Virtex-II block RAM (18K bit) can be configured in many widths. By picking a width of 18, three each, 10-bit components (Y’ or Cr or Cb), 1024 pixels deep, with two block RAMs can be accommodated. A system speed of 27 MHz on the input can easily be supported by even the slowest Virtex speed grades.

Whether buffering data from external memories or operating as Line Buffers, Virtex and Spartan-II block RAMs can handle the two focus frequencies for video. The SDTV rates of 13.5 MHz per pixel (27 MHz per digital component) and HDTV rates of 74.25 MHz (148.5 MHz per digital component) are well within reach for many of the Virtex and Spartan-II devices.

Reference Design

The reference design demonstrates two separate implementations of the two-line average, de-interlacing problem shown in Figure 6. The implementations require us to store a minimum of one line worth of information. The first implementation uses a SRL16 line buffer (Figure 7) while the second uses a block RAM line buffer (Figure 8). Both interface to a FIFO designed in dual-port CLB memory to cross the different pixel clock and memory clock domains. The output of the FIFO drives a ZBT frame buffer via a ZBT RAM controller. Other Xilinx reference designs are available for a DDR SDRAM and ZBT RAM.

Table 4 shows the results after “place and route” of the various modules implemented in this application note. All results were obtained using the Verilog versions of the designs with Xilinx ISE version 4.1i using XST as the synthesis tool. Results using the VHDL files are not shown, but are essentially identical. Virtex-II device results are for a –5 speed grade device. Spartan-II device results are for a –6 speed grade device.

<table>
<thead>
<tr>
<th>Design Name</th>
<th>Size LUTs/FFs</th>
<th>Speed Virtex-II Device</th>
<th>Speed Spartan-II Device</th>
<th>Ports</th>
<th>Power Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Reference Design Results

Conclusion

Line buffers for current video standards (HDTV and SDTV) are easily designed with the supporting block RAM or SRL16s in Xilinx FPGAs. The density of the block RAMs or SRL16 implementations support the number of pixels or pixel components per line. The speed of the block RAMs easily meet the performance requirements of the video standards. Line buffer to external frame buffer interfaces are made easier by small FIFOs implemented in Xilinx CLB, dual-port synchronous RAM. Demanding frame buffer interfaces can also be supported by Xilinx FPGAs as shown in the reference design with a ZBT interface or a DDR SDRAM interface.
References

1. Logic Devices Incorporated, LF3304 Dual Line Buffer/FIFO Data Sheet. 10/27/1999-LDS.3304-C.


3. The video standards beginning with ITU come from the International Telecommunication Union. ITU-R BT.656 and by ITU-R BT.601 standards are available on the International Telecommunication Union’s web site, http://www.itu.int/itudoc/itu-r/rec/bt/ for a small fee. The Society of Motion Picture and Television Engineers (SMPTE) standards are available on http://www.smpte.org for members or a fee.

4. Video Demystified, by Keith Jack, published by Harris, ISBN 1-878707-23-X, is a good beginners guide to video techniques. It can be read or purchased on line at the following URL: http://www.video-demystified.com


6. Charles Poynton, tel: +1 416 413 1377, fax: +1 416 413 1378, poynton@poynton.com www.inforamp.net/~poynton

Revision History

The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>05/21/02</td>
<td>1.0</td>
<td>Initial Xilinx release.</td>
</tr>
</tbody>
</table>