Accelerating DTP with Reconfigurable Computing Engines

Donald Macvicar[1] and Satnam Singh[2]

[1] Dept. Computing Science, The University of Glasgow, G12 8QQ, U.K.
donald@dcs.gla.ac.uk

[2] Xilinx Inc., San Jose, California 95124-3450, U.S.A.
Satnam.Singh@xilinx.com

Abstract. This paper describes the context in which a reconfigurable computing engine can be used to accelerate DTP printing functions. We show how PostScript rendering can be accelerated using a commercially available FPGA co-processor card. Our method relies on dynamically swapping in pre-computed circuits to accelerate the compute intensive portions of PostScript rendering e.g. drawing curves. We deploy a novel hardware description language which allows efficient utilisation of a limited FPGA resource. Experiments show that this technique could lead to a commerical viable FPGA co-processor for desk top publishing applications. We describe a run-time system for managing the FPGA and dynamic circuit resources. We also outline other compute intensive desk top publishing functions that can be addressed by FPGA technology.

1 Introduction

Imagine the office of a typical desk top publishing house: it will comprise of mid to high end PCs or Macintosh computers with large displays connected to a series of printers that can generate publication quality colour output at thousands of dots per inch (dpi). Rasterisation is the process of converting the high level page descriptions in PostScript into bitmaps that specify where to apply the ink on the paper. This is a very compute intensive operation and is often performed by a separate computer or *Raster Image Processor* (RIP) system. The applications that graphic designers run include image processing packages like Adobe Photoshop that perform compute intensive operations like convolution (e.g. Gaussian Blur) over huge true-colour images. Image processing and printing is often very slow.

Imagine taking an off-the-shelf FPGA PCI card and slotting it into these machines to accelerate typical desk top publishing functions. That vision is exactly what we are striving for in our projects at the University of Glasgow in conjunction with Xilinx Inc. This paper concentrates on how we are working to accelerate PostScript rendering with reconfigurable technology. Other projects have shown that it is possible to extend and accelerate the filtering operations of Adobe Photoshop using exactly the same hardware.

This paper introduces the desk top publishing market and describes the PostScript rendering technology that we are currently developing. This includes a collection of

rendering circuits that are dynamically swapped into FPGAs as and when required. It also includes a software architecture that manages the FPGAs. Rather than providing a stand alone rasterisation application, we show how this system can be used with the Windows operating system to systematically provide seamless support for hardware based PostScript rendering.

2 Desk Top Publishing

2.1 Document Creation and Preparation

Desk Top Publishing (DTP) is a rapidly growing area of the current computer application market. The increased use of high quality colour graphics has put an even bigger demand on the software and hardware used for DTP. The manipulation of a colour image to prepare it for printing requires a significant amount of compute power to perform the operation in a reasonable amount of time. The addition of instructions like those on the Intel MMX processors has also helped to improve the performance of image processing operations (e.g. filtering). The main bottleneck with these operations is the large amounts of data that have to be accessed to perform the operations.

2.2 Print Technology

Recent developments in print technology are having a significant effect on the whole printing industry. High quality printed material is now available in small quantities at reasonable cost. Digital print presses are making it possible and cost effective to produce short runs of high quality, full colour documents and also allow print on demand services to be offered. The internet has also had a major effect on the printing industry and the way in which documents are used. In the past a document was edited and printed before finally being distributed. It is now becoming more common to distribute documents in their digital form over some kind of network and print at the final destination.

Digital cameras and low cost high quality scanners along with increasing power and reducing cost of home and office computers has allowed many more people access to desktop publishing. Small companies and individuals can now produce high quality documents but require only small numbers of them thus the printing industry has had to change to meet these new demands.

The changes in the way that documents are being used and new print technology have forced changes in the pre-press production area. The original printing presses required each letter on a page to be positioned by hand on a large plate which was then coated in ink and pressed onto paper. The technology has evolved somewhat from those days but the principles are still the same. Computers are now used to generate the plates which are then mounted on cylinders and rolled over the paper thus increasing the speed of printing. Even using computers the cylindrical plates take time to produce and it is feasible only to use these techniques for large quantities. For colour printing four plates are required as a four colour process is used (cyan, magenta, yellow and black). The new generation of printers has reduced the cost and time taken to produce the plates used for printing thus making short runs possible.

The data used to create the plates used for imaging is a raster image of the page which is constructed from a digital description of the page. The most common language for page description is PostScript from Adobe Systems. Documents are created using applications such as PageMaker, FrameMaker or Quark Xpress. The document is then saved as a PostScript file which can be sent to a printer. Many print bureaus will accept PostScript files which they can print on the high resolution printers and imagesetters.

The process of converting the PostScript generated by the application into the bitmap data that is used by the print engine is a compute intensive one. The systems used for this conversion are called Raster Image Processors (RIP). RIP systems are often dedicated high performance workstations and cost tens of thousands of dollars. These expensive workstations are used solely for rendering PostScript which could be a waste of processing power if documents are not always being printed. Even these dedicated workstations can take hours to render documents and are thus often run in parallel with the printer, while one document is being printed the next is being rendered ready for printing.

There are two stages to rasterising PostScript. First the PostScript has to be interpreted to obtain the commands and associated data for each page in the document. In the second stage the data from the first stage is used to build a bitmap image of the page and is sent to the print engine. The page descriptions in PostScript are not page independent i.e. some data on the last page may be required on the first page. This interpage dependency makes it difficult to process pages concurrently.

For the FPGA-PS project PDF was chosen as the input page description language. PDF is very similar in power to PostScript but contains a static, page independent description of each page. Several packages are available for converting PostScript into PDF including Acrobat Distiller from Adobe Systems.

Adobe Systems Inc. have recently introduced two solutions for PostScript printing. Adobe PrintGear [2] is a custom processor designed specifically for executing the commands in the PostScript language. It is designed for desktop printers upto 600dpi and can produce approx. 8ppm which allows current print engines to operate at their full potential. The second solution is called PostScript Extreme [1], which is a parallel approach to PostScript RIPing. PostScript is first converted into PDF to get page independence and each page is then dispatch to one of upto ten RIP engines which produce the raster images for sending to the print engine. The current version of this system has been built for use with an IBM printer and costs an extra $30,000 on top of the $780,000 for the printer. This is based on the InfoPrint 4000 600dpi duplex system and is capable of producing 464 impressions per minute on a RIP while printing system. It is possible to output variable data PostScript pages with high quality graphics at these speeds.

3 FPGA Technology

In the case of the FPGA PostScript project the Xilinx XC6200 FPGA is used to accelerate the compute intensive areas of the PostScript rendering process. The FPGA is only utilised when there will be a clear advantage over using software. It is not prac-

tical or possible to implement the entire PostScript rendering process on a FPGA therefore only the areas that can benefit from acceleration are concentrated on.

Since space is limited on the FPGA, we use the discipline of virtual hardware to dynamically swap in circuits as they are required. Whenever possible, it helps to order rendering operations to avoid swapping, otherwise we might experience thrashing.

The following areas of the rendering process are being investigated for acceleration. These identified areas become more significant as the output resolution increases.

- **Matrix-Based coordinate transforms.** Every coordinate in a PDF file has to be multiplied by a six element matrix to obtain the absolute position on the page for the object.
- **Rasterisation of lines.** Converting a line from its specification as start and end points to the list of pixels on the particular line.
- **Rasterisation of Bézier curves.** Curves can be rendered by approximating the curve with a series of straight lines, which can then be rendered by the line rasterisation circuit.
- **Rasterisation of TrueType fonts.** TrueType fonts are specified as a series of curves thus rasterising fonts can use the Bézier circuit from above. This may prove too data intensive for FPGA implementation.
- **Anti-Aliasing.** When lines and curves are rendered onto a raster grid they may appear jaggy and uneven in intensity; anti-aliasing corrects these problems.
- **Colour correction.** Different raster display devices have different characteristics with respect to colour so when high quality output is required colours have to be adjusted for the particular output device.
- **Bitmap Compression.** Since PostScript (level 2) can process LZW compressed data the intention is to transmit an LZW compressed bitmap to the printer thus reducing the about of data transfer required.
- **RAM Bitmap to SCSI output.** The bitmap for a page is finally sent to the printer using DMA transfers over a high speed SCSI bus prevents this from becoming a significant bottle neck in the process.

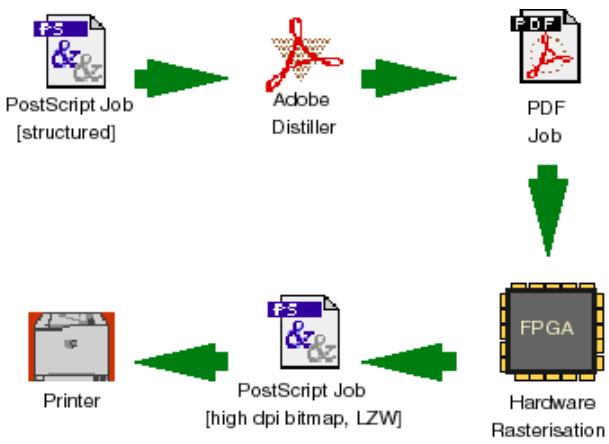


Figure 1 FPGA-PS System.

The FPGA-PS system uses several stages to perform the complete PostScript rendering process. The input PostScript job is first converted into PDF using Adobe Acrobat Distiller. This application is invoked through its control interface (OLE) which allows other applications running under Windows to communicate with Distiller and request it to perform work for them.

The PDF is then parsed to obtain the list of graphical objects that require rendering for a particular page. Currently only a small subset of the possible page marking operations are handled by the system. Only line and arc drawing operations are handled by the initial prototype because these were noticed as being those which could benefit most from a FPGA implementation which is geared towards technical drawings.

When the list of objects has been received from the parser they are rendered to the screen using Windows graphics drawing functions. The image displayed on screen can then be rendered for printing. The rendering process for printing uses a combination of hardware and software. The hardware FPGA circuits are only used to carry out tasks when it will benefit the entire system to do so.

The output of the hardware/software rendering process is a bitmap PostScript file which requires very little processing by the printer.

4 Case Studies

Several of the circuits for the PostScript rendering process have been implemented and integrated into the software system described above.

4.1 Line Rendering Circuit.

Scan conversion of lines to obtain the detailed information of which pixels are on a raster grid is a common and well known problem with many solutions. In PostScript there is a command for drawing a line given its end points. Since the initial prototype for the FPGA-PS system is aimed at technical style drawings scan conversion of lines is a useful algorithm to implement in FPGA.

There are many algorithms that could have been used but after reflection Bresenham's scan conversion algorithm 8. is particularly suited to FPGA implementation for several reasons. Firstly Bresenhams algorithm involves only simple integer arithmetic and has minimal initalisation. The algorithm also generates one co-ordinate pair per iteration of the main loop which simplifies the control circuitry necessary for the FPGA implementation.

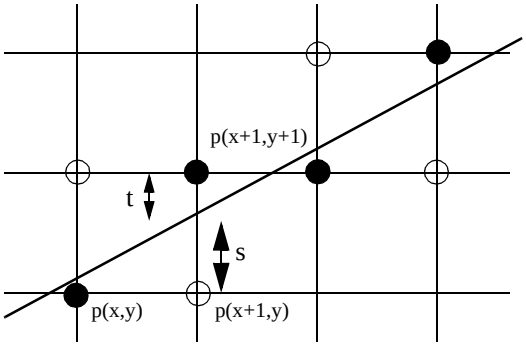


Figure 2 Scan Converting Straight Lines.

Bresenham's line algorithm is an incremental algorithm for scan converting straight lines. The pixels on a particular line defined by its end points are calculated by stepping along the axis with the largest difference between the end points. At each step *p(x,y)* either the pixel *p(x+1,y)* or *p(x+1,y+1)* is selected depending on the larger of *s* and *t*.

The code given bellow is for rendering lines with a gradient of between 0 and 1 but has been generalised for all lines in the circuit implementation.

```
Bresenhams (int x1,y1, x2,y2, colour)
{
// Initialisaton Section.
   int dx   = abs( x2-x1 );
   int dy   = abs( y2-y1 );
   int d = 2 * dy - dx;
   int inc1 = 2 * dy;
   int inc2 = 2 * (dy - dx);
// Assume that x1,y1 is start point.
   int x = x1;
   int y = y1;
```

```
    int xend = x2;

    WRITE_PIXEL(x, y, colour);
    while ( x < xend ) {
       x = x + 1;
       if (d < 0 )
          d = d + inc1;
       else {
          y = y + 1;
          d = d + inc2;
       }
       WRITE_PIXEL( x, y, colour);
    }
}
```

The implementation of the generic line drawing algorithm uses only three simple circuits. The x and y terms are calculated using a circuit which performs one of three functions depending on the control inputs given. The circuit takes a value and can either increment or decrement it and also can do nothing returning the original value. The error term d is calculated by adding one of two constants onto the previous value. The appropriate constant is select by multiplexors and is then added to the previous value with the result being clocked into a register.

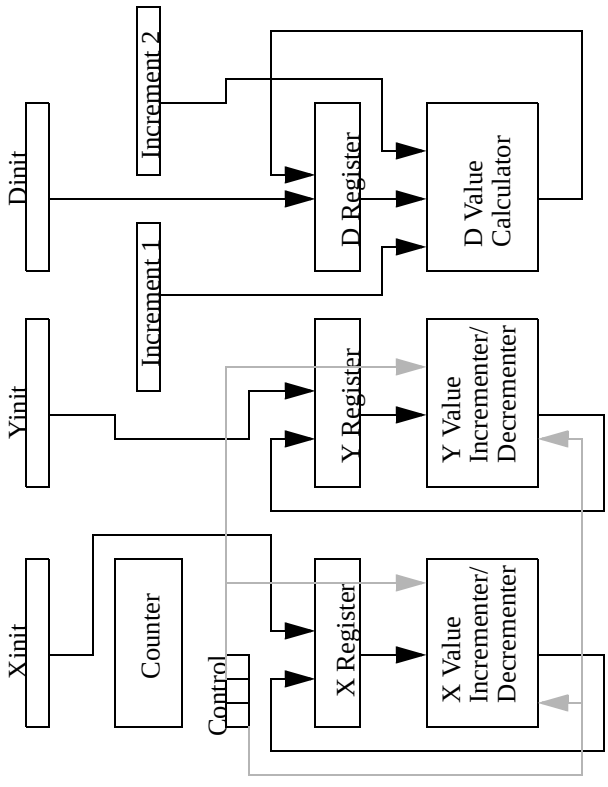


Figure 3 Line Scan Circuit Block Diagram

The circuits used to build the calculation units for each of the variables in the loop are built up from a bit level circuit and are parameterised on the size of the final circuit thus making a circuit that uses 16 bit integers, rather than the 8 bits as shown, is simple.

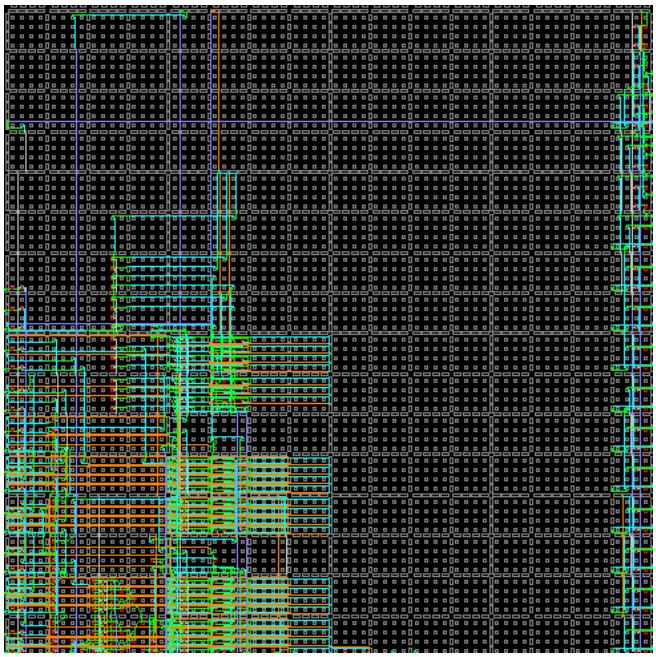


Figure 4 Line Rendering Circuit.

The above circuit produces approximately 10 million pixels per second. This could be improved by using look ahead carry in the adders rather than the simple ripple carry that is currently used. Other line scan algorithms can render two pixels per loop with similar amount of calculation to Bresenhams algorithm. Run-Slicing could also be used to speed up the process, if the line is split into two sections these could then be rendered in parallel thus giving double the performance but with the added cost in term of chip area used and the increased control logic. Having two circuits rendering in parallel would greatly increase the complexity of the interface to the SRAM.

4.2 Circle Rendering Circuit

The algorithm used to render circles is very similar to that used for straight lines. An error term is used to decide in which direction the next pixel on the curve is.

The error term used for the circle rendering is more complex to calculate than that for the line drawing algorithm. The error term is calculated by adding one or other of two values to the previous value but unlike the line algorithm the values are not constants and are dependant on the previous values of x, y and d.

$d_{n+1} = d_n + 2*x_n + 3;$
$d_{n+1} = d_n + 2*(x_n - y_n) + 5;$

The symmetry of a circle is used by this algorithm to reduce the number of calculations required. Assuming a circle centred on the origin it is possible to calculate the coordinates for only a 45° segment of the circle and use these

points to obtain the complete circle. At each stage through the loop in the code sample below the point *(x,y)* is calculated. The `WRITE_CIRCLE` function then takes this point and can plot the following list of points:

(x,y), (y,x), (y,-x), (x,-y), (-x,-y), (-y,-x), (-y,x), (-x,y)

```
midpoint_circle ( int radius, colour )
{
  int x = 0;
  int y = radius;
  int d = 1-radius;

  WRITE_CIRCLE(x, y, colour);

  while (x < y)
  {
    if (d<0)
      d = d + 2*x + 3;
    else
    {
      d = d + 2*(x-y) + 5;
      y = y - 1;
    }
    x = x - 1;
    WRITE_CIRCLE( x, y, colour);
   }
}
```

When implemented on an FPGA the calculation of the error term d takes the longest time because of the arithmetic required. The commutivity of addition and subtraction can be used to improve the speed of the calculations. For example in the else clause of the if statement the calculation requires three arithmetic operations but parallelism can be used to reduce time taken to that of two operations in sequence. The x-y part and the d+5 can be carried out in parallel before adding the two result to arrive at the final answer.

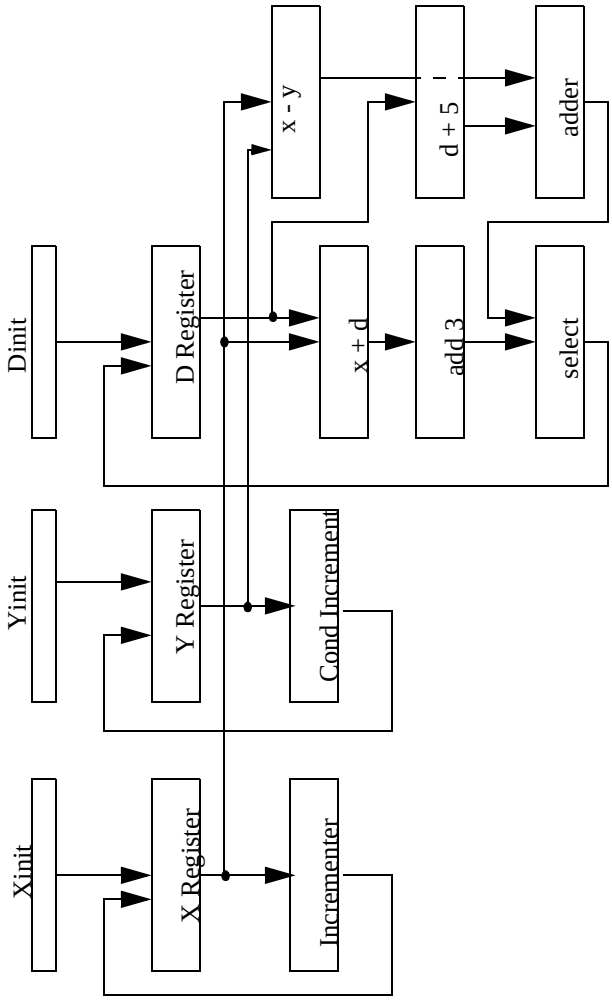


Figure 5 Circle Rendering Circuit Block Diagram.

We have also reported the design of a sphere rendering circuit realised on the XC4000 devices which delivers good performance [6]. It is interesting to note that the circle circuit is a special case of the sphere circuit (the sphere circuit was described in structural VHDL).

4.3 Bézier Curves.

Polylines are a first degree, piecewise linear approximations to curves resulting in large numbers of lines for a curve that is not also piecewise linear. In PostScript and PDF curves are specified using parametric cubic curves cf. $Q(t) = [x(t) y(t)]$

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x,$$
$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y, (0 \le t \le 1)$$

Curves can then be defined in terms of control points which are substituted for a,b,c,d in the above equations.

The general technique is to approximate the curve with a number of straight line segments but there are several ways to find the end points of the lines. Firstly the parametric equations can be evaluated for discrete values of *t* and then a straight line drawn between the resulting points. This process is very slow as it requires a large amount of floating point calculations. Also the curve is approximated by a series of equal straight lines which for some curves my be inefficient as they could have longer straight sections. A more efficient method of rendering the curve is do recursivly divide the curve in two and check each half for straightness. If it is not straight then divide again otherwise a straight line can be drawn.This provides more flexability as the straightness can be check to some tolerance resulting in a trade off between curve smoothness and the time taken to produce it.

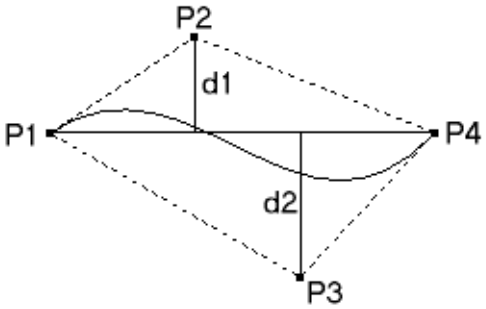


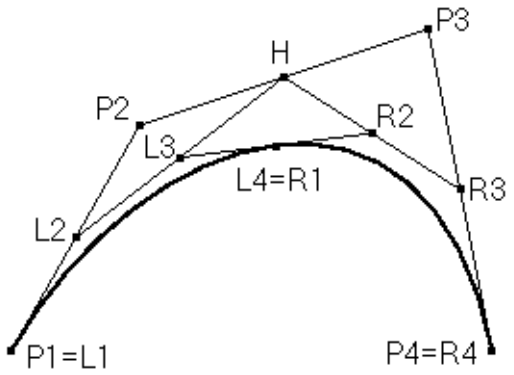
Figure 6 Bézier Curve and Control Points

There are two methods by which the division can be carried out. Firstly the parametric equations can be used to produce point on the curve at mid point of two previous points and then the straightness check involves checking for that the three point lie on a straight line. This is still very compute intensive as it requires constant evaluation of the parametric equations. The second method, which is used in our system, involves dividing the curve into two new equal curves. The curve is checked for straightness and only divided if it is not approximately straight. The test for straightness involves calculating *d1* and *d2* in Figure 6 and comparing them to some tolerance value. The calculation is a complex floating point one involving squares and square

roots. Since the straightness test requires a large amount of calculation a fixed depth of recursion is used with the depth of the recursion is dependant on the approximate length of the curve. Although this method involves more iterations of the recursion it does reduce the calculations involved.

```
recursiveBezier(curve, recDepth)
{
  if ( recursionDepth > 0 )
  {
     splitCurve(curve,leftCurve,rightCurve);
     recursiveBezier(leftCurve,recDepth-1);
     recursiveBezier(rightCurve, recDepth-1);
  }
  else
     drawLine(curve);
}
```

The distance between P1,P2,P3,P4 is used as a pessimistic estimate to the length of the curve. The distance P1,L4,P4 in Figure 7 is used as an optimistic estimate of the length of the curve. The logarithm too the base two of each of the above lengths is found. The depth of recursion is then set too the average of the two logarithm values.



L2=(P1+P2)/2, H=(P2+P3)/2, L3=(L2+H)/2,
R3=(P3+P4)/2, R2=(H+R3)/2, L4=R1=(L3+R2)/2

Figure 7 Splitting a Bézier Curve.

Figure 7 shows how a curve (P_1, P_2, P_3, P_4) can be split into two curves (L_1, L_2, L_3, L_4) and (R_1, R_2, R_3, R_4) which represent the left and right halves of the curve respectively.

A circuit for dividing Bézier curves in half has been designed and built using only integer arithmetic which is adequate. Improved results could be obtained by using fixed point real arithmetic. To perform the division of a curve two identical circuits are used one for the *x* components and one for the *y* components. The block diagram for one of the components is shown in Figure 8. Before rendering begins, all Bézier

curves are converted into a sequence of straight lines then the resulting lines are rendered with the others in the document. Currently software is used to do handle the recursion with the FPGA circuit being used to simply divide the curves. It is hoped that once the memory access and other problems with the VCC Hotworks board are corrected that the entire process of producing the straight line to approximate the curve can be done using a circuit on the FPGA with the resulting line segments written to the on board SRAM.

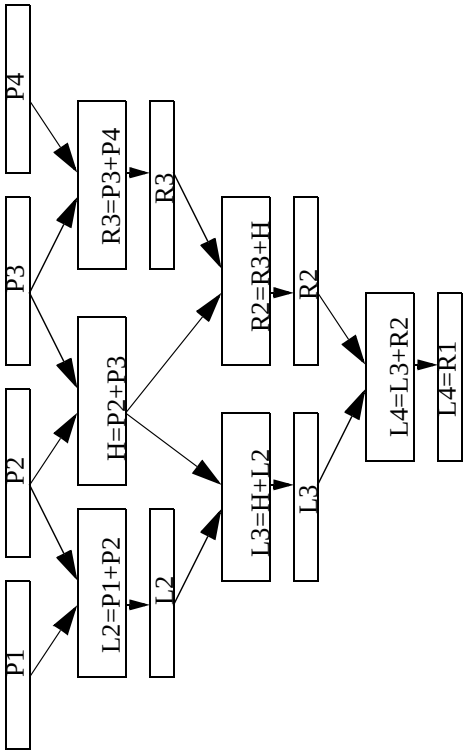


Figure 8 Block Diagram of Bézier Dividing Circuit.

All the circuits were implemented in an hardware description language called *Lava*, which is a variant of the relational algebraic hardware description language Ruby [5][7]. A key feature of this language is that it provides circuits combinators that encode circuit behaviour and layout. This allows us to specify circuit topology without explicitly calculating the co-ordinate of each cell. This in turn allows us to generate circuits which are far more likely to route in a reasonable amount of time.

5 Board Issues

We used the HOT Works boards from VCC which have Xilinx XC6216 FPGAs. This board is fine for development work but would cause excessive overheads in a com-

plete system. The job of rendering PostScript as well as being compute intensive also requires large amounts of memory. An A4 page (similar in size to a US letter page) in full colour at 600dpi requires approximately 32MB of memory. It would be impractical to have a whole page bitmap in memory at once so the bitmap must be paged from main memory and perhaps even disk for high resolution. The current PCI card has only 2Mb of SRAM which is enough for only a small region of a page and will thus incur much swapping if more memory was available then the swapping could be reduced.

Several other boards are available which may be better suited to this particular task. The SBX+ from SUN is one possible solution for several reasons. Current RIP systems are often based upon high powered workstations like DEC Alpha's or SUN Sparc Stations, since the SBX is designed for this architecture then many companies could benefit from the acceleration with minimal outlay. The SBX is also designed to be extendable using a daughter card with 1-4 FPGA/ASICs thus allowing a configuration that is suited to an individuals need. The FPGA chips that are connected to the base card do not have to be the same e.g. Xilinx and Altera parts can be connected to the same board. The use of different parts facilitates the use of different FPGA architectures which would allow algorithms to be implemented on the architecture which best suits it. LZW compression requires access to large look-up tables therefore using a LUT based FPGA to implement this would achieve better performance than using a fine grained FPGA.

The ACEcard from TSI-TeleSys has two XC6264 parts and a 100MHz microSPARC IIep processor core on one PCI card. The memory on this card although DRAM is fitted via a SIMM socket thus allowing as much memory as possible to be connected to the FPGA. Each of the FPGA chips also has a small amount of SRAM connected to it which could be used for storing circuits and thus reducing the swapping overhead. Having a processor and FPGA on the same board would allow for extra parallelism in that each FPGA could be doing one job and the processor another. The processor on the card be used to run normal code or can be used to configure the FPGA chips. Using the local processor and local memory would reduce the cost of swapping circuits onto the FPGA and would thus speed up the entire process.

6 Software Issues

The driver model in Windows NT is a multi-level one with different levels having different operating environments. There are several layers to a print driver. The system described above could be implemented at several different levels. At the top level is the section of the driver that an application calls. It would be possible to implement the FPGA circuits at this level but would mean that the application printing would be tied up for the time it took for the pages to be rendered. The rendered data is also far larger than the PostScript and this method would thus involve more network traffic. If the printer is to be used in a network situation then it is best for the FPGA rendering to be carried out at the lowest level or in the print spooler level. If the implementation is at the device interface level then a different driver would be required for different printers. The optimum level for implementing the FPGA-PostScript is thus at the print

spooler level. The print spooler simply takes the data from the application level driver and sends it down too the device level driver using a buffer to handle multiple print jobs. The spooler could thus convert the structured PostScript into bitmap PostScript before sending to the printer.

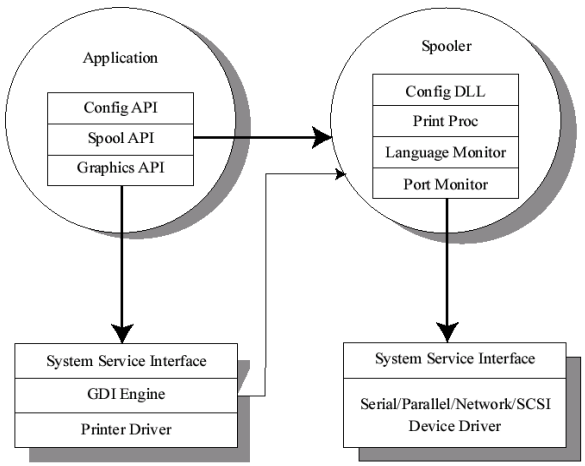


Figure 9 Window NT Printing Architecture.

Many printers can interpret the PostScript internally but this can be a very slow process. The FPGA system tries to remove as much of the processing as possible resulting in sending a PostScript document that has only a bitmap for the current page which needs no further processing.

When using the FPGA as virtual hardware the order in which the rendering operations are done becomes critical to the system performance. If many different circuits are required one after the other then the overhead of swapping in the new circuits will outweigh the acceleration. The graphical objects have to be very carefully ordered to minimise the swapping of virtual circuits. If objects on a page overlap then the order in which they are rendered will have a profound effect on the final result, the order in which the overlapping objects are rendered must be the same as that in the original PostScript file.

The memory required to hold a bitmap image of even one A4 page at current printer resolutions will range from 32Mb at 600dpi up to 2Gb at 5000dpi. Since it is impractical to have physical memory of that capacity especially in SRAM connected to the FPGA some form of paging system will be required. The paging of memory will also require careful ordering of the graphical objects to reduce swapping overhead incurred by reconfiguring the FPGA for different rendering operations.

A related project is developing a run-time circuit specialisation technique called partial evaluation. This technique is based on performing constant propagation at run-time, giving a systematic way to perform dynamic synthesis in a controlled fashion.

These techniques could be employed to improve the speed of certain circuits. In PDF every co-ordinate within a document has to be multiplied by a matrix called the Current Transformation Matrix (CTM). This CTM allows PDF to be device independent and is essentially used to scale the co-ordinates to the correct resolution for the current output device. During the processing of a page the values in the CTM remain constant for long periods of time. It would therefore be beneficial to customise a generic matrix multiplication circuit for the current CTM. This would require some analysis or prediction on the number of calculations the CTM will remain unchanged for.

Many publishing applications have plug-in technology such as Photoshop, Ilustrator and many other products even CAD tools. The circuit to perform a given function can be download to a general FPGA accelerator card allowing the same piece of hardware to be used to accelerate as many functions as possible. Due to the partially reconfigurable nature of the Xilinx XC6200 it may even be possible for more than one application to have circuits processing on the FPGA at the same time.

7 Performance

Using a test document which contains 15,768 lines totalling 92,262 pixels at a resolution of 72dpi the speed of the FPGA line drawing circuit was analysed. We used GSView32 from Russel Land of Aladdin running on a 150Mhz Pentium with 64Mb of memory to estimate the time taken for a typical software based PostScript rasteriser. This application takes approximately 5s to recalculate and display the image on screen after it has been read from the file. The speed of GSView32 is dependant on many factors other than just the amount of data to be rendered which include overheads from the poor performance of the graphical interface of the operating system. Using a straight forward implementation of Bresenham's line scan algorithm in C++ which simply renders the lines into a bitmap in memory it was measured to take approximately 1.73seconds to render the 15,768 lines at 72dpi in the same test document. Assuming that the same image is to be rendered using the FPGA at a resolution of 1000dpi resulting in approximately 93262*(1000/72) = 1,295,305 pixels must be rendered. The circuit can render at 5,000,000 pixels per second (using 16-bit arithmetic) thus it takes 0.26s to render at 1000dpi. This speed could be improved by utilising some of the techniques mentioned earlier. The transfer of the data for each line will also affect the total rendering time.

One of the severest limitations of our system is the very low performance of the PCI interface. Using one of the earlier VCC Hotworks boards, we have measured a transfer rate of just 0.7Mb/s, but the theoretical limit for PCI is 132Mb/s. VCC have recently improved the interface, but the bandwidth is still far too low compared to the speed at which the FPGA circuits rasterise the final image. In the future, we plan to investigate using Intel's Accelerated Graphics Port (AGP) [4] system allowing us to rapidly transfer the image over this dedicated bus (up to 533Mb/s), leaving the PCI bus for control signals.

We shall use the measured data transfer rate of 0.7Mb/s for writing to registers on the XC6200. There are 15768 lines in the image and each line requires 10 bytes of data - since we have a 32-bit bus this requires 3 transfers over the PCI bus per line since we

need to communicate 12 bytes per line. This gives a total of 12*15768=189216 bytes resulting in a transfer time of 0.25s to complete all data transfers. These two values added together result in a total rendering time of approximately 0.51s, which is significantly faster than the software.

8 Accelerating Image Processing

PostScript rendering is just one of many desk top publishing functions that are suitable for hardware based acceleration. We have also developed plug-ins for Adobe Photoshop which use the same VCC XC6200 Hotworks board to accelerate image processing operations like colour space conversion and image convolution (e.g. Gaussian Blur). Some of these filters operate at around 20 million pixels per second on the board, but as stated earlier the poor performance on the PCI interface on the card delivers a much poorer performance to the user. However, all the hardware filters that we developed still ran several times faster than their software only versions.

9 Summary

In summary, we report that we are getting closer to our vision of a desk top publishing studio exploiting dynamically reconfigurable technology for commercial advantage. We have designed and implemented some of the most important circuits required for PostScript rendering. We have developed the methodology of virtual hardware allowing us to swap in circuits as required into the FPGA. We are developing a run-time system to manage the FPGA board resources and to present a high level interface to the application layer. And finally, we have investigated where in the Windows 95 and NT architecture would be the best place to install the software that takes PostScript and dispatches it to the FPGA for rendering.

The main barriers at the moment include the unsuitability of the VCC Hotworks board for our applications. In the next stage of the project, we will move to a board that either has a superior PCI interface, or one that has an alternative channel for communicating the image (e.g. AGP). We also need far more image memory on the card, which might require us to move to DRAM instead of continuing with SRAM based cards. The TSI-TeleSys cards are a likely system for us to investigate. They would allow us to cache enough circuits on the board to accelerate swapping virtual circuits. They also have a faster PCI interface (although we have not tested this board ourselves).

In the recent IEEE Computer article *Seeking Solutions in Configurable* Computing [2] it was reported that ‘no companies are known to use reconfigurable computing for a competitive advantage.’ We believe that this project will eventually address this problem by providing a low cost alternative to expensive software and hardware that is required for rasterising PostScript.

The authors acknowledge the assistance of Dr. John Patterson with Bézier curve rendering. This work is supported by a grant from the UK’s EPSRC.

References

1. Adobe Systems. *Adobe PostScript Extreme White Paper*. Adobe System Inc. 1997
2. Adobe Systems. *Adobe PrintGear Technology Backgrounder*. Adobe Systems Inc. 1997
3. William H. Mangione-Smith, Brad Hutchings, David Andrews, André DeHon, Carl Ebeling, Reiner Hartenstein, Oskar Mencer, John Morris, Kirshna Palem, Viktor K. Prasanna, Henk A. E. Spaanenburg. *Seeking Solutions in Configurable Computing*. IEEE Computer, December, Vol. 30, No. 12. December 1997.
4. Intel. *Accelerated Graphics Port Interface Specification Revision 2.0*. December 11, 1997.
5. M. Sheeran, G. Jones. *Circuit Design in Ruby*. Formal Methods for VLSI Design, J. Stanstrup, North Holland, 1992.
6. Satnam Singh and Pierre Bellec. *Virtual Hardware for Graphics Applications using FPGAs*. FCCM'94. IEEE Computer Society, 1994.
7. Satnam Singh. Architectural Descriptions for FPGA Circuits. FCCM'95. IEEE Computer Society. 1995.
8. J.D. Foley, A. Van Dam. *Computer Graphics: Principles and Practice.* Addison Wesley. 1997.
9. Xilinx. *XC6200 FPGA Family Data Sheet*. Xilinx Inc. 1995.