Hardware Accelerated Text Display

Soheil Servati Beiragh

University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
https://scholar.uwindsor.ca/etd/4886

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000 ext. 3208.
Hardware Accelerated Text Display

By

Soheil Servati Beiragh

A Dissertation
Submitted to the Faculty of Graduate Studies through
The Department of Electrical and Computer Engineering in Partial Fulfillment
of the Requirements for the Degree of Doctor of Philosophy at the
University of Windsor

Windsor, Ontario, Canada

2013
© 2013 Soheil Servati Beiraghi

All Rights Reserved. No Part of this document may be reproduced, stored or otherwise retained in a retrieval system or transmitted in any form, on any medium by any means without prior written permission of the author.
Hardware Accelerated Text Display

By

Soheil Servati Beiraghi

APPROVED BY:

______________________________________________  Dr. N. Dimopoulos, External Examiner
University of Victoria

______________________________________________  Dr. I. Ahmad
Computer Science

______________________________________________  Dr. M. Ahmadi
Electrical and Computer Engineering

______________________________________________  Dr. R. Rashidzadeh
Electrical and Computer Engineering

______________________________________________  Dr. R. Muscedere, Advisor
Electrical and Computer Engineering

May 1st 2013
Declaration of Originality

I hereby certify that I am the sole author of this dissertation and that no part of this dissertation has been published or submitted for publication.

I certify that, to the best of my knowledge, my dissertation does not infringe upon any one's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my dissertation, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my dissertation and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my dissertation, including any final revisions, as approved by my dissertation committee and the Graduate Studies office, and that this dissertation has not been submitted for a higher degree to any other University or Institution.
Abstract

Web browsers and e-book are some of the most dominant applications on mobile devices today. They spend a significant amount of time handling text in these documents. Based on the experimental results from different commercial web browsers, the majority of the time spent to display text is dedicated to layout design and painting the bitmaps of the character glyphs on the screen; the time needed to rasterize the bitmaps of these glyphs is negligible. Many efforts have been made in software to improve the performance of text layout and display and very few are trying to come up with parallel processing schemes for System-On-Chip (SoC) designs to better handle this graphic processing. This work introduces a new novel hardware-software hybrid algorithm which performs the layout design of text and displays it faster by using a small piece of hardware which can easily be added to the SoCs of today’s mobile devices. This work also introduces a novel method for applying kerning to layout design process. The performance of the algorithms are compared to WebKit, the most widely used web rendering framework, and has resulted in a 29X and 192X performance increases in layout design when kerning is both used and not used respectively.
To My Selfless Mother,

My Kind Sister,

And the great soul of My Father,
Acknowledgements

I would like to thank my supervisor, Dr. Roberto Muscedere for his continued help and support during my entire time as a graduate student. I am very grateful for all that I have learned from him and his patience toward me.

Also I would like to thank my committee members, Dr. N. Dimopoulos, Dr. I. Ahmad, Dr. M. Ahmadi and Dr. R. Rashidzadeh for attending my seminars and their constructive comments.

A very special thank to Dr. M. Ahmadi for supporting me in every step of my life as graduate student.

In addition, I would also like to thank my parents and my kind sister for their life time support for me.
# Table of Contents

Declaration of Originality ......................................................................................... iv

Abstract ....................................................................................................................... v

Dedication ...................................................................................................................... viii

Acknowledgements ...................................................................................................... vii

List of Tables ............................................................................................................... xiii

List of Figures .............................................................................................................. xiv

List of Appendices ....................................................................................................... xvi

List of Abbreviations ................................................................................................. xvii

Chapter 1: Introduction ............................................................................................... 1

1.1 Motivation .............................................................................................................. 1

1.2 Evolution of the Text Display Process ................................................................. 5

1.3 Dissertation Objective .......................................................................................... 7

1.4 Dissertation Organization ..................................................................................... 8

Chapter 2: Background ............................................................................................... 9

2.1 Proportional Text Display Process ....................................................................... 9

2.1.1 Glyph Handling ............................................................................................... 10

2.1.2 Aliasing ........................................................................................................... 12
2.1.3 Kerning ........................................................................................................ 12
2.1.4 Layout ........................................................................................................... 18
2.1.5 Painting the Bitmap ...................................................................................... 19
2.1.6 Transfer to the Frame Buffer ......................................................................... 20
2.2 Other Research in Literature ........................................................................... 20
2.3 Methods to Improve Mobile Performance ......................................................... 22
  2.3.1 Simplify Content prior to Processing .......................................................... 23
  2.3.2 Optimize the rendering engine for limited scenarios .................................... 23
  2.3.3 Optimized algorithm or technique in the software rendering engine .......... 24
2.4 Summary ........................................................................................................... 24

Chapter 3: Analysis .................................................................................................. 26
  3.1 Introduction ..................................................................................................... 26
  3.2 Initial Testing and Platform Selection ............................................................... 26
  3.3 Hardware Platform Selection .......................................................................... 29
    3.3.1 Method of Measure .................................................................................. 33
  3.4 Basis of Analysis and Comparison (WebKit) .................................................... 34
    3.4.1 Internals of WebKit .................................................................................. 36
    3.4.2 WebKit on Microblaze ............................................................................ 39
    3.4.3 Performance Evaluation of WebKit .......................................................... 40
Chapter 4: Design and Development ................................................................. 43

4.1 Introduction .................................................................................................. 43

4.2 Software Only Implementation .................................................................... 43

4.2.1 Glyph Processing .................................................................................... 44

4.2.2 Layout Design .......................................................................................... 45

4.2.3 Placement .................................................................................................. 48

4.2.4 Transfer to the Frame Buffer .................................................................... 50

4.2.5 Performance of the Software Only Implementation ............................... 52

4.3 Hardware Design .......................................................................................... 52

4.3.1 Memory Interfacing and Hardware Control ............................................ 53

4.3.2 Hardware-Functions ................................................................................ 57

4.3.3 Application Programming Interface ....................................................... 60

4.4 Summary ....................................................................................................... 62

Chapter 5: Results .............................................................................................. 63

5.1 Introduction .................................................................................................. 63

5.2 Raw Engine Performance ............................................................................ 63

5.3 WebKit Engine Performance ........................................................................ 65

5.4 Effect of Burst Memory Access ..................................................................... 67
Appendix C: Text Display Engine IP Core ................................................................. 116

C.1 dispaymem2.vhd .......................................................................................... 116

C.2 user_logic.vhd ............................................................................................ 123

Appendix D: Sample API Code ........................................................................ 141

Vita Auctoris ...................................................................................................... 154
List of Tables

Table 3-1: Timing Comparison of Developed Software Engine between Intel and ARM Processors ................................................................................................................. 28

Table 3-2: Summary of components of XUPV5-LX110T Evaluation Board ............... 31

Table 3-3: Glyph Rasterizing and Layout Design in WebKit for a passage of text with one million characters ......................................................................................... 40

Table 4-1: BUS Interfaces available on XUPV5 ................................................................ 56

Table 5-1: Raw software timing results with no kerning .................................................. 64

Table 5-2: Raw software hardware hybrid timing results with no kerning ....................... 64

Table 5-3: Performance of proposed engine with and without Visual Kerning for a 2-million character novel ......................................................................................... 65

Table 5-4: Performance comparison between proposed engine and WebKit in Layout Design .................................................................................................................. 66

Table 5-5: A timing comparison for Bitmap Placement Phase between WebKit and proposed engine ............................................................................................................ 67

Table 5-6: Hardware resources required by the custom hardware .................................... 70
List of Figures

Figure 1-1: Samsung ARM Cortex A15 Exynos®5 System on Chip [3] .......................... 4

Figure 2-1 Glyph Metrics ............................................................................................................. 11

Figure 2-2: Comparing the effect of kerning on the placement of characters ................ 14

Figure 2-3: Effect of Kerning on Readability. Part (a) is output of Microsoft Word without Kerning and Part (b) is output based on the proposed method. ...... 14

Figure 2-4: Visual Kerning Calculation .................................................................................. 16

Figure 2-5: Measured distance between bitmap and bounding box edges ......................... 18

Figure 2-6: The average time spent for each step of the Text Display Process ........... 22

Figure 3-1: Beagle Board [22] ................................................................................................. 27

Figure 3-2: XUPV5-LX110T Evaluation Board. Image Copyright Xilinx© ....................... 31

Figure 3-3: High level structure of WebKit text rendering engine .................................... 37

Figure 4-1: Text Display Process ............................................................................................. 44

Figure 4-2: Layers of a Linux based Memory in an Embedded System ......................... 54

Figure 4-3: Multi Port Memory Controller Module and Interface options for XUPV5 .. 55

Figure 4-4: Block Diagram of the Final Hardware System ................................................ 59

Figure 5-1: Effect of Burst Access .............................................................................................. 73

Figure A-1: Peripherals and their parameters for the base system ................................. 82

Figure A-2: How to configure MPMC Step 1 ................................................................. 91

Figure A-3: How to configure MPMC Step 2 ................................................................. 92

Figure A-4: How to configure MPMC Step 3 ................................................................. 92
Figure A-5: How to configure MPMC Step 4 ................................................................. 93
Figure A-6: How to configure MPMC Step 5 ................................................................. 93
Figure A-7: How to connect ports of the developed peripheral.................................... 94
Figure A-8: IO Mapped Memory Addresses of the system............................................. 95
Figure A-9: How to configure Petalinux ......................................................................... 96
Figure A-10: Making necessary changes to include Xilinx Frame Buffer ....................... 99
List of Appendices

Appendix A: System Design Procedure .......................................................... 80
Appendix B: System Design Codes .................................................................. 101
Appendix C: Text Display Engine IP Core......................................................... 116
Appendix D: Sample API Code ........................................................................ 141
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DDR-RAM</td>
<td>Double Data Rate Random Access Memory</td>
</tr>
<tr>
<td>DFB</td>
<td>Direct Frame Buffer</td>
</tr>
<tr>
<td>DMA</td>
<td>Direct Memory Access</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>DTS</td>
<td>Device Tree Source</td>
</tr>
<tr>
<td>EDK</td>
<td>Embedded Development Kit</td>
</tr>
<tr>
<td>FB</td>
<td>Frame Buffer</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphic Processing Unit</td>
</tr>
<tr>
<td>HDMI</td>
<td>High Definition Multimedia Interface</td>
</tr>
<tr>
<td>IO</td>
<td>Input / Output</td>
</tr>
<tr>
<td>IP-Core</td>
<td>Intellectual Property Core</td>
</tr>
<tr>
<td>JPEG</td>
<td>Joint Photographic Experts Group</td>
</tr>
<tr>
<td>LCD</td>
<td>Liquid Crystal Display</td>
</tr>
<tr>
<td>LPDDR</td>
<td>Low Power-DDR</td>
</tr>
<tr>
<td>LTE</td>
<td>Long Term Evolution</td>
</tr>
<tr>
<td>MMU</td>
<td>Memory Management Unit</td>
</tr>
<tr>
<td>MPEG</td>
<td>Moving Picture Experts Group</td>
</tr>
<tr>
<td>MPMC</td>
<td>Multi-Port Memory Controller</td>
</tr>
<tr>
<td>NPI</td>
<td>Native Port Interface</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PDA</td>
<td>Personal Digital Assistant</td>
</tr>
<tr>
<td>PDF</td>
<td>Portable Document Format</td>
</tr>
<tr>
<td>PLB</td>
<td>Processor Local Bus</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>SoC</td>
<td>System on Chip</td>
</tr>
<tr>
<td>TFT</td>
<td>Thin Film Transistor</td>
</tr>
<tr>
<td>TTF</td>
<td>True Type Font</td>
</tr>
<tr>
<td>ULP</td>
<td>Ultra Low Power</td>
</tr>
<tr>
<td>VHDL</td>
<td>VHSIC Hardware Description Language</td>
</tr>
<tr>
<td>VHSIC</td>
<td>Very High Speed Integrated Circuit</td>
</tr>
<tr>
<td>XPS</td>
<td>Xilinx Platform Studio</td>
</tr>
</tbody>
</table>
Chapter 1
Introduction

1.1 Motivation

Many different types of multimedia content are available today on computer systems but still text and reading materials remained as the dominant content, whether on commercial desktop PCs or small handheld devices [1]. Despite the dominance of text content, a significant amount of effort has been made to enhance the user experience with 3D contents and media streaming [2] but very little work has taken place in the area of text rendering and display to make the text rendering engines faster.

Since the evolution of the personal computers, CPUs have become faster and more capable but still users are not satisfied with their experience. A good example of this situation is what one can experience with new cutting edge tablet computers. A high end tablet can easily stream a Full High Definition 1080p video with no lag and even send it out through an HDMI port to a large screen TV, but if you try to scroll through a long text based webpage or an E-Book, the device cannot keep up with your page flipping speed and it will start showing blank pages until you stop and let it catch up.

It is clear to any person familiar with computing systems that video and image handling is a much more complicated task for CPUs than 2D text display and this can lead one to
the conclusion that faster CPUs with more cores might not be very useful in fulfilling some basic tasks. The reason behind such a short coming is that although the text rendering process is fairly simple, it involves many none-cachable memory accesses and poorly predicted conditional branches, which makes it a time consuming task. In most of today devices, there are several co-processing units to help the main processor when dealing with 3D contents or media streaming, but there is no special unit to help the CPU with tasks like text rendering because they are considered easy algorithms which should be handled by the CPU on its own.

All handheld devices have restrictions in power and physical size. As the technology advances, companies build Cellphones, Tablets or E-Readers with higher speed CPUs, thinner sizes while trying to maintain longer lasting batteries. When Multi Core CPUs where introduced to the mobile industry, the main issue that came up was the power consumption. Add to this their higher and higher operating frequencies as well as peripherals such as high resolution/luminance displays, and high speed LTE Mobile access, will further increase the challenges for battery designers. Although companies are all trying to build more efficient displays, and radios, the demand for power keeps increasing.

As anyone might have experienced, although the technology on cellphones has advanced during the past few years, they have become far more demanding on power. For example, although doubling the number of cores in the CPU will double the number
of transistors, and potentially double the power consumption, consumers don’t necessarily experience double the response times from their devices. This is generally due to the fact that most of the applications available don’t even take advantage of the multicore system. It is a difficult and complicated task to develop applications that use multi-threading or can run on multiple cores in parallel.

To mitigate some of the high power requirements, today’s devices implement different policies for power management to reduce the clock speed when it is not required, but still as soon as the clock speed drops the user experiences longer delays from the system and most users prefer to keep charging their devices instead of operating at lower clock speeds.

This dilemma forces manufacturers to search for other solutions than simply using higher clocked CPUs. A good example is the media co-processors (audio and video decoding) that exist in most of the SoCs in mobile devices. Such a unit works in parallel with the CPU and requires much less power which improves the user experience by letting the CPU perform more useful tasks.

Many of today’s low power computing devices like E-Readers, Tablets, Cellphones and even some of the newer Laptops use SoCs in their architectures and advertised as being highly computational capable based only on having strong CPU performance.
An SoC is a processing unit which houses the main CPU of the computer and several other co-processors and peripherals in a single silicon die. This design analogy is aimed to build small, low power, and high speed computer systems. In most of today’s commercial SoCs, a CPU is embedded alongside a number of GPUs (i.e. ULP GeForce cores in the NVIDIA Tegra 4), a video acceleration unit, cache memory, and some radio and DSP controllers. The combination of GPUs and a cache memory makes the user experience much smoother for Multimedia contents and 3D graphics. The above Figure 1-1 shows SAMSUNG Exynos 5 which is an example of the latest SoCs being used in mobile devices today.
1.2 Evolution of the Text Display Process

The text mode in the early generations of IBM Personal Computer had the advantage of lower memory consumption and very fast screen updates. Text screens were fixed sizes (either 80 or 132 columns with 25 or more rows) and dedicated text display hardware which read the contents of these small arrays which contained the ASCII value of the text to be shown. The hardware would cross reference the desired character to a pre-rendered mono-spaced glyph and showed the image on the screen. The hardware was able to update the display such that it met the vertical refresh time of the desired resolution. These first generation display systems of personal computers did not have the rich graphics which we are accustomed to today and because of that, there were very limited choices for fonts and other graphical decorations. This original form of text mode can still be seen today as it is enabled during the PCs power on stage. Changes to the system configuration are done in this mode as the firmware of the PC is quite small compared to the OSs which they eventually boot into. As video hardware improved, developers soon started using the full graphics options which allowed them to create custom environments and proportional spaced fonts which replicated the output of high quality laser printers. However the wide range of hardware and vendors made it difficult for the software developers to use as each device had to be programmed separately. There was little support for all the custom resolutions however a standard known as the Video Electronics Standards Association (VESA) BIOS extensions allowed some flexibility for this [4].
With the advent of Microsoft Windows, software drivers provided by these hardware manufacturers eliminated this programming problem and offered software developers an API interface independent of the hardware that was easy to work with. A wide range of vector based fonts were becoming available which generated very rich documents unlike pre-rasterized fonts of the past. The demands of these developers were quickly squashed as these software APIs were slow due to the amount of code required to manage the “windows” as well as operate the displays. Hardware designers soon started adding in acceleration which improved the user experience; operations such as fast fills and bit blitters for scrolling the display quickly. Over time, more advanced features have been added as transistors have become smaller and fabrication processes have become faster.

Today’s PC entry level graphics hardware can perform media acceleration as well as high quality 3D images. The improvement in CPU performance is the sole reason software developers have the ability to further enrich our computer interfaces with endless eye-popping effects. Text display to this day however has seen no benefit of hardware acceleration. Today’s high resolution displays with very high dots-per-inch (DPI) demand significantly more resources to render text compare to the lower resolution displays of the past. Processor and network speeds can easily access and decode information, however the time to display said information is slow and often leads to delays which the user quickly notices. As this software is ported to mobile devices where computing and memory resources are much more restrictive, the delays are far more noticeable.
The process of displaying proportional text is not algorithmically complicated however it is memory intensive as it manipulates large amounts of arbitrary, none-cachable data with many non-predictable branches.

1.3 Dissertation Objective

As can be seen in the example shown in Figure 1-1, modern mobile processing units are built as SoC. This architecture makes it possible to add co-processing units that can have interaction with the main processor with higher speed buses which reduce any loss in time or latency. These modern SoCs all contain media decoding hardware units, and camera image processors, as well as 2D and 3D graphic accelerators. The addition of any other components should not be challenging to the designers.

This dissertation will present a novel hardware/software hybrid engine which can accelerate the text display on embedded systems with restricted resources. The design is intended to be added as a co-processor to the existing SoC environment.

This dissertation performs an in depth investigation on the text display process to determine the bottle necks and short comings of algorithms and methods in use by consumer electronics software developers. These results are used to guide the development of a novel solution to make the user experience more smooth and fluid by eliminating the delays which have become easily noticeable on the mobile platform.
1. Introduction

In order to validate the effectiveness of the proposed method, the developer of this work will compare the performance of the final product with a standard software engine that is being used in today’s consumer electronics devices.

1.4 Dissertation Organization

Following this chapter, this dissertation will continue with a description of the text display process in Chapter 2 as well as literature search. Chapter 3 will perform a series of analyses of existing software engines to determine the performance bottlenecks; it will also cover the hardware platforms used for this evaluation. In chapter 4 briefly details the hardware development process and some of the challenges; a full explanation is found in the appendices. Chapter 5 shows the performance comparison of the proposed engine and the software competitors. Lastly chapter 6 will contain the conclusion and recommendations for future steps.
Chapter 2
Background

This chapter will thoroughly examine the process of displaying proportional text in a modern computer graphics environment. Non-proportional fonts and the legacy “text mode” operation will not be discussed any further as the main focus of this dissertation is on text display in modern GUIs.

Additionally, other research literature is reviewed which suggests text placement time could be improved.

2.1 Proportional Text Display Process

The text display process is normally consisted of few separate tasks:

- Glyph Handling (Extracting character glyphs metrics and rasterization),
- Process any special feature like kerning distances or other decorations,
- Plan the layout of a text passage based on a text box or screen size,
- Paint the bitmap of the text passage by copying the individual rasterized glyphs to a surface bitmap,
- Transfer the final surface bitmap to the frame buffer to be shown on a display.
2. Background

2.1.1 Glyph Handling

The first step in any software that displays proportional text is to process the font file that is used for the text object. A PDF reader or a web browser, for example, first looks at the font name specified in the source document file and reads the applicable font file (usually on a local disk). Different fonts are part of different font families and the software decides which font file to use based on the family of the font mentioned in the source document. It is sometimes the case that the font needed is not present, so software is given translation tables to select the most appropriate font. This may result in poor document rendering so PDF documents may contain embedded fonts and similarly web pages can contain embedded fonts known as web fonts [5]. The most common used type of font files today are either True Type Font (.TTF) or Open Type Font (.OTF, a superset of TTF) which are supported in almost all computer operating systems. Software to decode TTF and OTF is freely available, however some of the algorithms are patented, but work arounds have been developed which avoid the issue [6] [7].

Based on which operating system or rendering engine is being used, a font processing library will be called to deal with extracting data from the font file. In almost all GNU/Linux variant OSs the FreeType library is being used to process all font files including the legacy pre-rendered bitmaps. Microsoft Windows has a native TTF and OTF engine although FreeType has been ported and is used by many packages to maintain cross platform compatibility.
2. Background

A font file basically consists of a number of tables containing bitmap or vector data of characters, their metrics (properties such as height and width of each glyph), as well as other information pertaining to kerning and aliasing. A font processing library like FreeType reads this tabulated data from the font file and, based on the size requested by the calling software or OS, rasterizes the bitmaps of characters and calculates the metrics and returns them to a text rendering engine through a series of data structures. Figure 2-1 shows the glyph metrics extracted by WebKit from a font file.

![Figure 2-1 Glyph Metrics](image)

A font processing library like FreeType can rasterize bitmaps of glyphs based on the size and aliasing option requested by rendering engine. This work does not consider the bitmap rasterizing and font file processing as it will be shown to consume very little CPU time. The scope of this work is to introduce a solution for faster and more efficient text display mechanism with the glyph bitmaps as one of its inputs.
It is important to note that almost all rendering engines use some kind of caching mechanism for character bitmaps so that they can process text objects faster and not waste time and resources to process the font file repeatedly. FreeType itself does not offer a high level caching mechanism, whereas libraries (i.e. Pango) use FreeType for rendering and uses its own caching mechanism.

### 2.1.2 Aliasing

With the advancements in display technology for both higher resolutions and varying depths of illumination, traditional text display of black and white glyphs generated very high contrast images which affected readability. The solution to this was to smooth the edges of the text by blending the background colour with the foreground text \[8\] [9]. The result is known as text aliasing [10] [11]. The concept of aliasing is not new, however Microsoft developed a method known as cleartype [12] which exploits the nature of liquid crystal displays (LCD) so that they can further improve the smoothing over traditional text aliasing. Due to patents on cleartype, this will not be discussed here. FreeType offers basic text aliasing which we use in the resulting bitmaps of this work.

### 2.1.3 Kerning

The next step in the text display process is to calculate and apply any special features for character bitmaps. Some of these special features are simply different kinds of
2. Background

decorations (underline, shadows, strike-thrus, etc.), however some are more important in the readability experience of user.

OS developers and application designers consider a number of restrictions for the users which ultimately makes the performance of their software better. E-Book readers generally only support certain types of fonts and certain sizes for the text; which eliminates the need for vector based font support. Although advertised as powerful processing units that can handle even more tasks than simply reading books, E-Book readers suffer from large delays when the user changes the size of the text or the font.

Kerning, an important factor to improve readability of displayed text [13], is an example of one of the features that are ignored in text rendering. Kerning, by definition, is the process of adjusting the spacing between characters in a word or a phrase to achieve a more readable result [14]. The designed rendering engine of this work has the ability to handle kerning without any significant performance loss comparing to the existing software based rendering engines.

The following example shows how kerning can affect the spacing between the two letters W and A.
2. Background

In Figure 2-2(a) _W_ and _A_ are placed with no kerning applied, which means the bounding box of character _A_ started immediately after the end of the bounding box of _W_. In (b) the bounding box of _A_ has overlapped with bounding box of _W_ since the _A_ can be pushed closer to _W_. The importance of kerning arises when we have a situation like _W_ and _A_ in a word. The words WAKE and Wake are shown with and without kerning. This situation happens for many other combinations of letters.

![Figure 2-2: Comparing the effect of kerning on the placement of characters](image)

In Figure 2-3: Effect of Kerning on Readability. Part (a) is output of Microsoft Word without Kerning and Part (b) is output based on the proposed method.
In the first set, the letter W looks to be disconnected from the word. This will lead to lower readability of the text. With kerning the second set shows words that feel more connected and easier to read.

There have been some efforts from software developers and font designers to come up with a kerning algorithm to make the displayed text more pleasant to read [15] [16]. These methods are mostly based on mathematical equations, and geometrical relationships of individual glyph curves. The TTF and OTF font files have the ability to provide kerning through the use of tables (described as combinations of pairs of characters), however the implementation is poor when using large font sizes. In most cases the data isn’t even provided at all. As one might have experienced, even with famous commercial text editing applications like Microsoft Word, none of these methods have made a significant difference in the appearance of text and especially for large sized texts where the situation can be very drastic.

Another approach to kerning is to use visual inspection; schemes similar to what humans do in hand writing. For example, we see that we can squeeze W and A more because their shapes allow us to without overlapping them onto each other.

The same approach can be used in by a computer by looking at the bitmaps of each characters and how close they can be placed on the screen beside each other. This method requires examining the bitmaps of the two letters row by row to determine what is the
least amount of spacing such that they won’t overlap. We call this method “visual kerning”.

![Figure 2-4: Visual Kerning Calculation](image)

In order to make the visual kerning more appealing, one must take into consideration the fact the pixels at the edges of most characters don’t have the full luminance due to aliasing; we call these sub pixels. If the minimum acceptable distance between two characters be considered as one pixel, there is difference in appearance of two pixels that are fully illuminated and separated by one empty pixel and other two that are partially illuminated and again separated by one empty pixel. Kerning does not have to consider sub pixels, however the results are far better but with the cost of more resources as it requires going through each of the bitmaps row by row and calculating the kerning distance for each pair.
2. Background

The author of this work has implemented both scenarios in the software version but in the final hardware version does not take sub pixels into consideration so that the final design is simpler.

The additional overhead of this method comes from two sets of calculations being added to the text display process. As noted before, FreeType extracts the properties of characters, the metrics, from the font file. Figure 2-1 shows some of these properties in relation to the glyph.

The width of each glyph is considered to be the width of the bounding box of the character bitmap. Also for many cases the advance (the horizontal increment to the next character) is either equal or larger than the width. As shown in Figure 2-2, in many occasions, the characters can be pushed even closer to each other than the suggested advancement. Therefore the information provided by the font renderer is not necessarily accurate when determining the kerning distance between a pair of characters.

In order to accurately measure the visual kerning distance between any pairs of characters, first the proposed engine needs to generate two more sets of data from the bitmaps of characters: The actual distance between the glyph bitmap and the bounding box for each row on the right and the left. Once two characters need to be placed, these measurements are compared on a row by row basis to determine how closely the bitmaps can be placed.
2. Background

![Figure 2-5: Measured distance between bitmap and bounding box edges](image)

It is possible to cache the kerning distances for different character pairs, but this may not be a good choice since so many possible combinations can exist due to the number of characters that exist in font files today. Unicode offers up to 32-bits to reference a character; the goal being to generate single font files for all languages. Any use of caching would require strict frequency rules and memory management; a task for software, not hardware.

One of the advantages of the proposed hardware-software hybrid text display engine is to make it possible to implement features like visual kerning in a way that the resulting experience is still faster than traditional software methods.

### 2.1.4 Layout

After the necessary character bitmaps are extracted from the font file and feature calculations are performed, the text rendering engine starts the layout design process for the text based on the screen properties obtained from the OS or GUI system.
Text layout is not a complicated task for CPU but involves many memory access, additions, and comparisons. The software keeps adding the width of characters until it reaches the end of the available width of the screen and goes to the next line. The computer considers the alignment (e.g. left, right, center, justified) and calculates where each line should be started on the screen.

This process seems to be a simple task for high speed CPUs, but because of the degree of software abstraction today, it can easily become a time consuming task. The kerning and layout calculation is actually the process of determining the individual characters destination address in RAM.

This algorithm must also determine the height of the line for each row based on the height of each of the characters and uses it in the placement phase to calculate the final size of the bitmap for the desired text.

### 2.1.5 Painting the Bitmap

After the layout design step, the rendering engine will perform the placement by copying bitmaps of characters to calculated places in the rendering surface. This process involves reading bitmaps of letters pixel by pixel from an address in RAM and writing them into another address in RAM. Although it is simple algorithmically, the process requires considerable CPU time as all the data fetches are effectively non-cachable due to
the large amount of data being copied. Furthermore, the looping strategy is short and prone to poor branch prediction causing many pipeline flushes.

### 2.1.6 Transfer to the Frame Buffer

Once the painting on the surface is complete, the engine passes the surface, via some method, to a frame buffer. This method is based on the computer system being used. Modern GPUs, for example, will use this surface along with others and composite them together by either simple overlays or possibly using more elaborate transparencies; this is done by hardware. The process of transferring the surface can be impeded by restrictions and limitations of the operating. Frame buffer memory space is considered a privileged area and a “driver” is necessary to guarantee simultaneous access is controlled.

The proposed design uses direct frame buffer access to place the surfaces directly on the display screen. This decision is valid as the memory space in most embedded devices is shared between the main CPU and GPU. This method guarantees the maximum transfer speed from surface to frame buffer which is normally done via software using basic memory move opcodes.

### 2.2 Other Research in Literature

The 2D hardware in mobile SoCs has traditionally been used by the user interface for scrolling and image scaling. Recently some software companies like Google started to
use the capabilities of the GPUs to perform image compositing as well as processing of vertex graphics [17]. In fact most desktop web browsers today have some form of GPU acceleration (usually compositing) [18]. It is reasonable to assume that this desktop code will migrate into the mobile devices in time.

The process of displaying normal text is considered as a 2D graphics operation. All aspects of this task are being done solely by the CPU. One of the major users of text display engine in modern computer devices are Web Browsers. All major Web Browsers have their own graphic rendering engine that reads the DOM tree (Document Object Model) of a web page, renders the page, and paints it to the surface to be displayed on the screen. Decoding different embedded contents such as images or interactive Flash are performed through external libraries or plugins. The rendering engine of a web browser does the layout design of the page and also renders the text.

Benchmarking of Microsoft Internet Explorer [19] and Apple Safari [20] shows that anywhere from 40-70% of the time spent by the application to display the web page is dedicated to calculating the layout design. Multimedia elements like Flash contents or images are not the main concern in layout design. Text objects are the main bottleneck of layout calculations. In the process of displaying text, each glyph will be rasterized to a bitmap image which are much smaller in size in comparison to normal inline images or Flash contents. The information in [19] and [20] is reproduced in Figure 2-6 using their worst case results.
2. Background

This timing analysis illustrates that decoding font files and rasterizing glyph bitmaps are not major contributors to the CPU load whereas up to 90% of the time is spent for the layout design and the placement of bitmaps on the surface. This result is not surprising as glyph bitmap caching can reduce the load on the CPU whereas copying these individual bitmaps over and over can contribute significantly more. Our own results, shown later in Chapter 3, will corroborate this result.

2.3 Methods to Improve Mobile Performance

Several techniques are being used in today’s devices in order to enhance the general performance.
2. Background

2.3.1 Simplify Content prior to Processing

Some mobile browsers use special proxy servers to either reduce or pre-render complex pages prior to sending them back to the mobile device. Although an elegant solution, it places the burden of computation on server farms as well as significantly reduces the privacy of the user.

Many content providers offer multiple versions of their website for different types of platforms and web browsers. In such instances a Blackberry user would see a different webpage as compare to an Apple user.

Another trend is that native applications are being developed for each platform to eliminate the processing delays of browsers entirely. Although speed and reduced bandwidth are significant advantages, developing websites and clients for each platform places a burden on the site developers.

2.3.2 Optimize the rendering engine for limited scenarios

Embedded OS and application developers usually limit the choices for users in order to increase performance (speed, power, etc.). In many cases the device only supports a hand full of font styles and has limitations on image and media size and formats (e.g. video profiles).
2. Background

2.3.3 Optimized algorithm or technique in the software rendering engine

Application developers, OS kernel designers and companies who manufacture mobile processors always try to come up with techniques and algorithms to better utilize the processing power of devices. Implementing video decoders, 3D graphics accelerators, and other co-processing units inside SoCs is the major effort by manufacturers to solve performance issues of CPUs. There are also software designers who try to implement software parallel processing schemes to existing sequential algorithms [21]. One of these efforts is the one described in [20] which will attack bottle necks of rendering engines in web browser applications. However it is a well-known fact that “Y” number of cores does not equal “Y” times improvement in performance as communication and synchronization overhead can reduce performance.

2.4 Summary

This chapter covered the general process of displaying proportional text on modern UI systems. There are four major steps: glyph handling, layout, placement, and transfer to frame buffer. Other evidence is shown which suggests that the speed of layout and placement of modern mobile browsers is very intensive and could be improve to yield
better performance. Also, some methods are listed which are employed by mobile devices to improve performance, although some hinder usability.
Chapter 3
Analysis

3.1 Introduction

Prior to any decision making in the design of the proposed system, an in-depth investigation of the text display process and a timing analysis for each step is performed. This chapter aims to determine the bottlenecks in the text display process and what solutions are available to make the process faster.

3.2 Initial Testing and Platform Selection

Initial tests were performed by creating custom software on a desktop PC in the GNU/Linux OS which placed glyphs rendered by FreeType in a large bitmap surface which is saved to disk. This code performed the fundamental algorithm discussed in Chapter 2: glyph rendering via FreeType, and text layout (with kerning) and placement utilizing glyph bitmap caching. These tests showed that the PC’s performance was far exceeding what was expected. Modern PCs utilize far more resources to improve performance compared to their mobile counterparts. A more suitable testing platform was needed.
3. Analysis

The ARM architecture is currently the dominant one in the mobile market. Therefore, in the interest of fair comparisons, the proposed system should be targeted to work in such an environment. The Beagle Board [22], an embedded system board with similar hardware as those found in mobile devices at the time, was selected. As shown in Figure 3-1, the Beagle Board is an embedded system based on a Texas Instruments OMAP 3530 ARM Cortex A8 superscalar processor. The SoC of this board houses 128MB of LPDDR RAM and 256MB NAND FLASH and a 3D graphic accelerator. Other necessary peripherals to run a full GNU/Linux OS are also available on this board.

![Beagle Board Diagram](image)

Figure 3-1: Beagle Board [22]

The same GNU/Linux distribution (Ubuntu) [23] was compiled and installed for this system. Two comparisons were performed, the first to compare this code to a well-known
3. Analysis

tool for rendering text to bitmaps, PDF2TIFF, and the second to compare processor architectural performance. The later used a vintage equally clocked desktop PC with similar RAM speeds. The results are shown below in Table 3-1. Neither of the code used rely on graphics or any other co-processors; they are simple tools which generate a bitmap output.

Table 3-1: Timing Comparison of Developed Software Engine between Intel and ARM Processors

<table>
<thead>
<tr>
<th></th>
<th>600MHz Celeron PC</th>
<th>600MHz Beagle Board with ARM Cortex A8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Number of Characters (Letters)</td>
<td>3200</td>
<td>3200</td>
</tr>
<tr>
<td>DPI</td>
<td>600</td>
<td>600</td>
</tr>
<tr>
<td>Proposed Code Time (Kerning Calculation + Placement – any adjustments)</td>
<td>0.3 Second</td>
<td>1.84 Second</td>
</tr>
<tr>
<td>PDF2TIFF Time (Placement – any adjustments)</td>
<td>1.7 Second</td>
<td>37 Seconds</td>
</tr>
</tbody>
</table>

The table shows that both the proposed software and PDF2TIFF perform significantly faster on the Intel x86 processor as compared to the ARM (the timing is for “user time” only, no OS overhead). The power savings in ARM based processors in comparison to Intel x86 processors mostly comes from the difference in the architecture design [24]. The simpler architecture design of ARM processors leads to smaller micro-instructions set which requires more clock cycles than Intel x86 processors to execute the same algorithm resulting in the significant performance difference. The Intel x86 processor also has the distinct advantage of additional on-chip cache. The difference between the
3. Analysis

The proposed code and PDF2TIFF illustrates the impact of code abstraction and the need for code optimization. PDF2TIFF was given a text only document, but its pure text performance was poor as it was never intended to operate on the ARM architecture.

As can be seen in the results, two processors with same clock cycle perform much different based on their architecture design. This shows the necessity of having optimized methods when dealing with embedded processing units of today’s mobile devices.

The results in the above table also seem to indicate that the glyph rasterization process provided by FreeType is not very time consuming compared to the task of placement. Another result shown later in this chapter will further show this is the case.

3.3 Hardware Platform Selection

The Beagle Board is a good platform for testing software implementations, but any custom hardware is impossible to include into this system as there is no interfacing points into the SoC. There was no environment which existed at the time this work began which offered an ARM processor with a programmable hardware fabric. In March 2011, during the mid-development of this work, Xilinx announced the Zynq platform which is an ARM processor linked with an FPGA fabric. This platform would have been ideal unfortunately it wasn’t until the beginning of 2012 that devices and software were easily available. It was the decision to maintain the current development on the environment of choice.
In order to fairly show comparisons between software only and the proposed design, an environment which was capable of running a modern OS as well as being hardware customizable needed to be found. The only choices at the time were FPGA based, and very few offered all the necessary key components. One of these major components is the OS. The intended use of the proposed hardware design is to be controlled by software (e.g. WebKit) which is designed for modern OSs. From a development point of view, the best choice is the GNU/Linux system which fits the needs for the software but also offers maximum flexibility as it is open source thus allowing for more in depth analysis and modifications. Any integration of custom hardware requires some type of driver; and the Linux kernel facilitates many ways to achieve this. Petalinux [25], a well maintained distribution of GNU/Linux for the Xilinx Microblaze soft-core processor, was the major contender as an evaluation environment. Altera offered their own variant of a soft-core processor, however there were no GNU/Linux builds available. A number of other features required were an environment with a reasonable amount of RAM, high resolution display output, and wired networking. Although Altera had hardware which met these needs, the OS was the key decision maker and the Xilinx hardware was selected.

The Xilinx XUPV5-LX110T Evaluation Platform FPGA Kit [26] was used as the hardware development environment for the proposed design. This board supports the soft core Microblaze architecture only (no PowerPC support) and has the required peripherals
3. Analysis

to build a complete embedded system capable of running the Petalinux GNU/Linux variant. Figure 3-2 shows the board and Table 3-2 summarizes its components.

![Figure 3-2: XUPV5-LX110T Evaluation Board. Image Copyright Xilinx®](image)

| Table 3-2: Summary of components of XUPV5-LX110T Evaluation Board |
|---|---|
| FPGA | Xilinx Virtex®5 XC5VLX110T FPGA |
| PROM | 2x Xilinx XCF32P Platform Flash PROMs (32 MB each) |
| Storage | Xilinx SystemACE™ Compact Flash configuration controller |
| Memory | 64-bit wide 256MB DDR2 small outline DIMM (SODIMM) module |
| On Board Flash | On-board 32-bit ZBT synchronous SRAM and Intel P30 Strata Flash |
| Timer | Programmable system clock generator |
| Audio | Stereo AC97 codec with line in, line out, headphone, microphone, and SPDIF |
| Display | 16x2 character LCD |
| | DVI Output and Controller |
| | VGA Output and Controller |
| Communication | 10/100/1000 tri-speed Ethernet PHY |
| | USB host and peripheral controllers |
| | RS-232 port |
| | JTAG programming interface |
3. Analysis

The hardware design process is done using Xilinx Platform Studio (XPS) v12.4. There are newer versions of XPS available but they were found not to be 100% compatible with various IP cores and the Petalinux tools. Modifying any of these components or tools was considered to be out of the scope of this work.

The Microblaze processor for this board can be clocked up to 125MHz, which is lower than the speeds seen on today’s mobile SoC processors. The intent of this work is not to compare the proposed system with a current mobile SoC, but to show an improvement through the use of additional hardware. In this case, the Microblaze processor and environment offer a platform to properly measure the performance of software versus hardware solutions. It can always be argued that the new high speed processor might not benefit from the proposed system, but no system was in place at the time to make this judgment.

Although this environment is not the ideal ARM based one as specified earlier, the software produced was all based in “C” and “C++” and contained no specific reference to the Microblaze architecture. The hardware interfacing is memory based which is typical for all co-processor and peripheral designs. The proposed design should therefore be easily ported to the new Xilinx software tools which support the Zynq platform.
3. Analysis

3.3.1 Method of Measure

The performance evaluations of this work are primarily for speed improvement; therefore all measurements are respect to time. Measuring time is generally not very accurate as the hardware timers in most system have a limited degree of precision. Software designers mainly use the kernel functions to access these timers for any type of comparison. In the case of the Beagle Board and the Xilinx Microblaze processors, the GNU/Linux system only provides a resolution down to a millisecond; which is not very fast for hardware performance measurements.

A simpler solution is to count the number of clock cycles during a particular operation. The real time can be derived from the number of clock cycles by using the following equation:

\[
\text{Time} (s) = \frac{\text{Number of Clock Cycles}}{\text{Clock Frequency of the System}} \quad \text{Eq. 3.1}
\]

By counting the number of clock cycles taken to execute a task, one can achieve the highest possible degree of accuracy in measuring time on a system. Also when number of clock cycles is used as the unit of measure the performance enhancement results will be independent from the clock speed of the system. Therefore no matter how fast the CPU is, the SoC can benefit from the clock cycles saved by using the proposed method.
In order to perform this measurement, a custom hardware timer is added to the Xilinx system and recorded before and after the interested function is executed. All of the timing results presented from this point onwards are derived from counting the clock cycles. These results are generated from several iterations of the same tasks with varying inputs averaged together.

3.4 Basis of Analysis and Comparison (WebKit)

It is crucial in any research work to find proper candidates for performance evaluation comparisons. There are several factors that must be taken into consideration when choosing the competitor, but one of the most important is that the competitor should be an industry standard. We need a competitor that is a trending solution and there are current efforts for evolution and optimization. It is also important to have a fair comparison and have sufficient information about the competitor, its properties, and limitations. These factors limit the choices for this work to ones that are open sourced so they can be examined in detail and, if possible, to extract detailed evaluation results.

In order to have a strong argument about the result of a work, one should compare the results with the state of the art technology being used. For this purpose it was decided to compare the final timing results of this work with a software engine that is being used in many of the today’s handheld devices. This will show the performance difference between a well-designed and well maintained software engine and the proposed. If the
proposed engine performs better than a software engine being used as a standard, then the work has a proven value.

Most mobile devices today use a GNU/Linux derivative OS (i.e. Android) [27]. Linux has the advantage of using open source libraries and applications. Web browsers and E-Book reader applications are the best environment to investigate the performance of the proposed text display engine. E-Book readers are very basic applications compared to web browsers. Web browsers deal with all types of media and interactive elements whereas E-Readers simply present text.

Almost all of the devices that operate on a GNU/Linux based OS have a native web browser or some third party ones that use the WebKit [28] engine. WebKit is a powerful and sophisticated web page rendering engine that handles the process of displaying a webpage from the start of reading data from a network connection up to the end result displayed page on the screen. Google Chrome, Apple Safari, BlackBerry Browser, Opera and many others use WebKit as the web page rendering engine. Teams of programmers from all different companies (such as the aforementioned ones) are contributing to WebKit to enhancing its performance. This makes WebKit an excellent candidate for comparison. It is interesting to note that Android itself uses parts of WebKit in its OS framework for UI components. Therefore any improvement in the WebKit system could impact the whole OS.
For this work WebKitDFB [29] [30] was used as it included an output system for Direct Frame Buffer (DFB) access. DFB access was required to show the output on the Microblaze hardware. Additionally, the same code base was used on a development PC so that the code could easily be inspected, debugged, and tested prior to testing on the Microblaze hardware.

3.4.1 Internals of WebKit

WebKit is a complete web page rendering engine which performs all the tasks needed in a browser to display a web page. It has many different components inside it that perform tasks like handling java elements, security features and much more. Part of WebKit is the rendering engine. One of major challenges of this work is to examine the steps WebKit takes to render and display elements of a web page. WebKit has a very large code base (approximately 1.4 GB) and requires about 24 hours and 6 GB of RAM to compile on an average modern desktop PC. For the development process, all WebKit code was cross compiled for the Microblaze hardware.

In order to have an accurate performance evaluation, it is important to find out the exact locations inside the code base where any task takes place. The code was compiled (debug build) and an in-depth investigation was performed on the code by using “DDD” a free GNU debugger application [31].
When WebKit loads a web page from internet, it splits the page to objects. Then, based on the type of each object and place of the object on the screen, it starts to render the object and copy the bitmap to the frame buffer.

In an HTML file, a text object is declared similar to the following example:

```html
<p style="text-align: center; margin-left: auto; margin-right: auto; color: #000000">
  <span style="font-family: Arial, Helvetica, sans-serif; font-size: 14pt">
    This is a Sample Text.
  </span>
</p>
```

After decoding the HTML file, WebKit reads the DOM tree objects extracted from the file. A text object in the DOM tree can contain a single or multiple paragraphs which in either case will be considered as a single array of text. WebKit reads the text array and its
style data and then it decides whether it needs to calculate the layout design or not. If needed, it starts to design the layout for the text object.

WebKit goes through the text array and based on the available width of the browsers window calculates where each line of text on the screen ends. In the meantime, based on the height of the characters in each line it calculates the vertical position of the next line on the screen. Throughout this process, WebKit breaks the text object to Render Block objects each representing a single line of text. The process of breaking the text object into Render Block objects and calculating the vertical position of them takes place in the “RenderBlockLineLayout.cpp”.

The layout design process is very similar to what is explained in chapter 2. WebKit reads each character, calls FreeType to rasterize the glyph bitmap, gets the advance for the character from FreeType, keeps adding them until it reaches the width of the screen, then it decides it has to go to the next line.

If the text array is larger than 8K characters, it first reads the text array in 8K chunks and places them in memory. Then it calculates the layout for the whole text based on the style information and screen properties. This means it will go through the whole text and breaks it into Render Blocks without considering if the whole text fits in the view window or not.
Other objects in the page will also be considered as Render Blocks and their horizontal and vertical positions on the screen will be calculated based on their width and height and their relative positions to other Render Block objects in the page.

After the layout design is done, WebKit starts to place the page on the surface. At this point it will start from the top of the page and paints each block on the surface until it reaches the height of the surface. Whenever the screen is scrolled, based on the change in the position of the view window by scrolling, WebKit decides which Render Blocks are now in the view window and places them on the surface.

This is a very simplistic and brief description of a complicated process that takes place inside the WebKit rendering engine whenever it renders a text object. This process involves many function calls inside WebKit and from other libraries.

### 3.4.2 WebKit on Microblaze

The Petalinux tools only create a system image that only contains only the minimum required libraries to boot up GNU/Linux on the Microblaze processor. In order to port WebKit to the Microblaze architecture, a number of other libraries must first be ported. This is done by using the Microblaze cross compiling tool chain provided as part of the Petalinux package and making many changes to the source codes of some libraries as needed to match the requirements and properties of a Microblaze processor. Following is
the list of libraries ported to Microblaze architecture in order to run WebKit on the Microblaze embedded system:

- Libcurl
- Libdirectfb
- Libenchant
- Libflex
- Libfontconfig
- Libfreetype2
- Libgcrypt
- Libglib2.0
- Libicu
- Libjpg
- Libleck
- Liblight
- Libpng
- Libsoup
- Libsqlite3
- Libxml2
- Libxslt
- LibWebKit

Although this build of WebKit uses DirectFB as the frame buffer controller, it still does not paint the page directly to the frame buffer. It uses a window handling library named Liblight and paints the webpage on a surface provided to it by Liblight.

### 3.4.3 Performance Evaluation of WebKit

Table 3-3 shows the comparison of the time taken to rasterize bitmaps of glyphs and the time taken by WebKit to design layout for a passage of text with one million characters.

<table>
<thead>
<tr>
<th>Font Size (Pixels)</th>
<th>Rasterizing (ms)</th>
<th>Layout Design (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>403</td>
<td>108,150</td>
</tr>
<tr>
<td>14</td>
<td>452</td>
<td>120,100</td>
</tr>
<tr>
<td>18</td>
<td>474</td>
<td>125,836</td>
</tr>
<tr>
<td>24</td>
<td>518</td>
<td>134,978</td>
</tr>
<tr>
<td>32</td>
<td>570</td>
<td>140,336</td>
</tr>
</tbody>
</table>
All of the performance evaluations of this work are performed by using public domain English literature novels with different lengths. The novel files are taken from [32] and are free to access.

The results shown above further confirm what is claimed earlier in this chapter from the tests on the Beagle Board. The rasterizing of the glyph bitmaps takes significantly less time than layout design for the text. FreeType is a modestly sized code base which decodes mostly all known font file formats. Any attempt to parallelize this code by use of special hardware would be unproductive as it already functions quite quickly on the intended Microblaze and ARM architectures. Based on this fact, the proposed design will only include the layout and placement phases for acceleration.

Due to the complexity in level of software abstraction in WebKit, the proposed design will be a hardware-software hybrid. The software portion will translate and prepare information from WebKit into the necessary context where then the hardware will process it. This approach is very similar to that taken with any add-on hardware as the software component is most commonly known as a “device driver” or device API (Application Programming Interface).

This design will be implemented and compared with a software only version to determine if it is in fact viable solution to improving the performance of text display in mobile devices. Chapter 4 will cover the specifics of this design from both the software and hardware aspects.
3. Analysis

3.5 Summary

This chapter performed numerous timing analyses on different softwares under different environments to determine where the bottlenecks in the proportional text display process lie. It was found that the glyph rendering process is relatively insignificant compared to the layout and placement tasks. It was decided that these two steps would be implemented in hardware to see the potential performance gains. The intended development platform, the Xilinx Microblaze system, was also introduced.
4.1 Introduction

This chapter will go through the process of designing the proposed hybrid hardware-software text rendering engine in more technical details. It will discuss the steps taken, decisions made and reasoning behind these choices.

There are always several different approaches to develop a hardware algorithm. The design environment and elements in this work are chosen based on the availability and cost considerations for an academic work and also the preference and expertise of the developer.

4.2 Software Only Implementation

The primary objective of this work is to make the process of text display faster and more fluid to minimize the update delays. The first step to achieve this goal is to design a very minimalistic algorithm which can be translated to hardware easily.

A software implementation offers an excellent model in order to determine the accuracy of the algorithm through comparisons to other software engines. Inefficiencies and special cases can easily be dealt with since software can be changed easily. The
software implementation is coded using C and the FreeType library is used as the glyph rendering processing engine. Apart from FreeType, the only other libraries used are the standard C found in all GNU/Linux distributions.

Figure 4-1 shows a flow diagram of the entire algorithm. The body of this dissertation only shows simple pseudo code detailing the major steps in the algorithm.

![Text Display Process Diagram](image)

**Figure 4-1: Text Display Process**

### 4.2.1 Glyph Processing

At the beginning of its process, the algorithm reads the name of the font file, size, style and text box properties from the either the source document (e.g. HTML file, Rich Text Document, etc.) or memory address.

A simple caching policy is used in this algorithm, cache everything. There are numerous existing works for caching mechanisms and strategies, which are out of the scope of this work. Since the written language of the text under test is English, it is
possible to cache all the letter, numbers, and symbols in under 128 entries. As shown in
Chapter 3, the time required to render the glyph bitmaps is negligible compared to the
layout and placement phases. Therefore, this policy is acceptable for the purposes of this
work. The algorithm, based on this policy, uses FreeType to render all the glyph bitmaps
and stores the necessary metrics needed for layout and placement into storage arrays.

Any kind of decoration or special feature should be considered from this point forward
in the rendering process. This work does not deal with applying decorations or other
features except for kerning.

If enabled, visual kerning data will be calculated from the rendered glyphs as described
in the second chapter. The minimum distances on the right and left side are calculated for
each character and stored in two arrays.

### 4.2.2 Layout Design

Since we are assuming a simple cache policy, all the necessary information is ready for
the layout process to begin. The layout process is necessary for two reasons: To setup
data so that the user may select and highlight text in the document, and to determine the
overall length of the document. WebKit actually performs two layouts, one with the
vertical scroll bar present on the side, the other without, if necessary. If the document is
small enough, the first layout will compute a vertical size smaller than that of the
paintable area. Should this happen, WebKit reworks the layout without the vertical scroll
bar so that it does not have to show it. This is an acceptable compromise since if the
document is small, both layouts shouldn’t take any significant amount of time to be
processed.

The data generated in this process is intended for use with WebKit. It requires four
arrays be populated during the layout process. This data will be used during the
placement step later.

**Character Advances Array**

Character Advance Array holds the relative horizontal position of each glyph bitmap on
a line of text. Without visual kerning, the position is generated by the advance metric
provided by FreeType. With visual kerning, the appropriate spacing between the current
and next character is computed by comparing the spacing between the glyph bitmaps and
their bounding boxes for each row. This is obviously a more complex operation and will
require more time to compute. It may be possible to speed this up if a dynamic cache is
created during the computation of each pair to save time for future identical pairs.

**End of Line Array**

Based on the available width of the screen or window on which the text should be
displayed, the algorithm determines on which character of the array of text in which the
line ends. This decision is made when the sum of the individual advances exceeds the
maximum width of the line. There are several considerations in the process for spaces at
the end of lines and blank spaces that happen based on the alignment at the beginning or the end of each line. The pseudo code details this more clearly.

**Line Height Array**

When the algorithm reaches an end of a line, it needs to determine where to start vertically placing characters of the next line. There are several policies to decide the height of lines in the design layout. Some rendering engines use the largest height of all characters bitmaps and this way all lines will have same uniform. Very few unconventional engines use the average which sometimes results in overlaps, while others calculate the line height for each line in order to better use the screen real estate.

This algorithm calculates the height of each line during the layout phase based on the tallest character (above and below the baseline) in the line. This will result in more efficient usage of the screen real estate and more consistency in paragraphs.

**Line Start Array**

Since the option exists for the type of horizontal alignment for text (e.g. left, right, centered, and fully justified), the starting point for each line must also be known. Based on this decision, the algorithm must set this array accordingly. The result for “left” and “fully justified” is always zero, where as the result for right and centered it is based on the space left per line.

The layout design process is described in the following pseudo code.
4. Design and Development

For the whole text array do the following,

Read the glyph index of the character,
Read the glyph index of the character next to it,
If the character is a space then
  Read the glyph index of the previous character,
  If the previous character is not space then
  Mark the current character as space_character;
else
  Set space_length equal to Width of the space glyph;
  Set word_length equal to 0;
  Count the number of spaces by using space_count = space_count + 1;
//Apply kerning calculation between current character and the next character,
Read the left kerning distance for the current character,
Read the Right kerning distance for the previous character,
Based on the vertical position of the two adjacent characters calculate the advance for
the previous character and move the cursor to the correct horizontal position,
// Kerning is applied,
Add the advance to the temporary length of the Line;
If temporary length of the Line + width of the next character is greater than Available
Width then
If current character is space then
  Line ends at current character;
  Set Line Length Index equal to Available Width - Line Length + space length;
else
  If next character is space then
    Line ends at next character;
    Set Line Length Index equal to Available Width - Line Length;
    And set Next Line Length Index equal to width of the space glyph;
  else
    Line ends at current character;
    Set Line Length Index is equal to
    Available Width - Line Length + word length + space length;
Now based on the alignment calculate the horizontal start position of the line,
If Alignment is Right then Line starts at Line Length Index;
If Alignment is Left then Line starts at the left most position on the screen;
If Alignment is Center then Line starts at (Line Length Index/2);
If Alignment is Justify then Line starts at the left most position on the screen;

4.2.3 Placement

The placement phase is essentially a series of memory movements from the cached
bitmaps to the destination “surface” bitmap. In this step the algorithm reads the character
from the text array to calculate the source address and reads the arrays built in the layout
4. Design and Development

phase to calculate the destination address. The glyph bitmap is then copied from the source to the destination. This operation, as noted earlier, is performed poorly on CPUs since the memory being copied is never cachable due to its large size and low frequency hit rates. Furthermore the data is copied row by row in small segments which cause poor branch prediction resulting in many pipeline flushes.

WebKit is smart about the amount of text to perform placement on; it is usually just twice the paintable screen area. When small screen changes are requested, such as a scroll down by 1 or 2 lines for example, the image is already painted. Once the screen is refreshed, it performs placement on newer data. This is a type of predictive caching.

The following pseudo code briefly describes the placement phase of the algorithm.

For the whole text array do the following,

If current character is space then
  If Alignment is justify then
    If Line Length Index mod space count equals to 0 then
      Adjust the width of space glyph by (Line Length Index / space count)
    Else
      Adjust the space glyph by
      (space number * Line Length Index / space count) - Sum of Adjustments;
  Else
    Adjust the space glyph by
    (space number * Line Length Index / space count)

  Set Adjustments Accumulation equal to Accumulation + the space glyph width adjustment;
  Set space number equal to space number + 1;

// Now paint the glyph bitmap on it calculated position on the surface,

For the current character do the following row by row
  Copy the bitmap of current character to destination;

Move the cursor forward by the calculated advance for the current character from the layout design step,

If End of Line is reached then
  Add the height of the previous line to the vertical position of the cursor;
  Set the horizontal position of the cursor for the start of the line based on the alignment;
4.2.4 Transfer to the Frame Buffer

In principle, the surface and frame buffer are the same thing, but exist in two different places. Modern CPUs offer virtual memory capabilities which modern OSs use to ensure system stability. This stability comes at the cost of performance. The processors in past mobile devices were essentially microcontrollers running very streamlined OSs with no memory and process protection; performance was relatively fast but stability was quite poor. The surface and frame buffer were the same thing in this case. With today’s devices, there is a clear difference between the two. The surface is located in “user space” while the frame buffer is located in “kernel space”. The kernel is responsible for maintaining the stability of the system, so it controls access to all devices which is especially useful in situations where multiple processes are attempting to use the same resource. The case is the same for the frame buffer, which is the content of what is to be shown on the screen. This is a linear portion of physical memory shared by the CPU and the display controller. The surface on the other hand is a linear portion of virtual memory in a user process (e.g. WebKit). Virtual memory is not guaranteed to be organized in a linear fashion, so the contents cannot be easily transferred from the surface to the frame buffer. The Linux kernel, by design, does not allow the user space to access the kernel space even if there are benefits in terms of performance to be gained. It is the ideology of the design that the two should never interact unless by software drivers so that stability is maintained. A significant performance increase could be obtained by allowing a device to
perform a DMA directly to user space, however this is not allowed. The mandated procedure is to DMA to kernel memory, and have the driver copy the result to user space; clearly inefficient from a memory access point of view. Unfortunately Linux on the mobile platform is still in its infancy and the kernel designers have failed to see the advantages of such interaction for this platform. The only solution is to perform memory moves between the two spaces to exchange data.

This whole point may be moot as modern web browsers are using the compositing engines found in GPUs to aid in the production of web content. Originally, for example, a background image on a web page would be drawn on the surface first, and then the text after. With compositing, two independent surfaces are combined by the GPU directly into the frame buffer thus requiring no work from the CPU.

In either case, at some point, the surface must be copied from the virtual memory into physical memory so that the GPU can access it. This is done by simple looping of memory moving CPU opcodes; or in “C” the “memcpy” function is usually optimized for the target platform.

For the proposed system, a software driver known as Direct Frame Buffer [30] is used to provide direct access to the frame buffer in user space. This is done by mapping the frame buffers physical space into the user space as a new memory address. Therefore the software has direct access to the device. However, the surface is still built in virtual
memory so in the end a copy is still performed which inherently reduces system performance.

4.2.5 Performance of the Software Only Implementation

The results of this implementation are shown in Chapter 5 alongside the hardware implementation results.

4.3 Hardware Design

Once the software model was shown to provide accurate results, its algorithm was then transferred into hardware. This was done by manually coding the algorithm in VHDL, a hardware description language. The hardware design is essentially a state machine which performs the necessary operations in sequence using the least amount of clock cycles as possible. The hardware implementation performs the necessary operations of layout and placement and the resulting image is placed directly in the frame buffer as the hardware has direct access to this memory region.

The largest challenge of this implementation is the integration into the existing Microblaze environment. The hardware is required to integrate with other components of the system to ensure that data is shared reliably between the components. The following
sections will address the challenges and decisions made so that this interfacing could be accomplished.

### 4.3.1 Memory Interfacing and Hardware Control

The communication link between custom hardware and software (Microblaze CPU) is a vital component in the proposed system and plays an important role in optimization. One of the first steps in developing the hardware system is the decision on the method to control the hardware and how to share data between the CPU and custom hardware.

Any memory management unit (MMU) based Linux kernel classifies the memory to three layers: user space, kernel space, and hardware resources. This layering mechanism is implemented in order to preserve valuable OS data from corruption and make the system consistent and reliable. Permission to access any layer of memory is governed by the kernel via the MMU and direct memory access (DMA) controller.

The top most layer in this architecture is the user space where normal applications are executed within and a typical user can interact with inputs and outputs of other applications. User applications can request dynamic memory allocations which will be virtually indexed by the kernel from physical memory.

The next layer is the kernel space where the system holds the OS code and variables which manage the operation of all user programs. This space is restricted to kernel
operations and device drivers enable the communication between to and from user space. They can also interact with physical memory or hardware.

The third layer down is the hardware resources which can be any combination of DDR2 RAM, Hardware Registers, or Ports. This is platform specific, and in this case pertains only to the Xilinx Microblaze system. All of these resources are directly accessible by the kernel via memory access as each resource is memory mapped.

Any device driver inside the kernel space can directly access a physical address which maps to either the DDR2 RAM or a Control Register. In the user space, direct access is not directly allowed but the kernel can map a virtual address to them via the “mmap” function and “/dev/mem” (see later).

DDR2 RAM access offers the ability to share large amounts of data with the custom hardware, but at the cost of facilitating the interface with the memory controller and the associate hardware as well as lengthy RAM latencies (will be detailed below). Whereas control registers only offer a small space for interface (several words) but the interfacing
is simpler. The impact of the complexity of the control register interface is noticed later as the decode logic becomes a key factor in the maximum clock speed of the system. The more register space used, the higher the chance the system won’t meet timing specifications.

The custom hardware component communicates with the RAM module through a memory controller unit which is directly connected to the DDR2 RAM. In the Microblaze system this interaction is implemented through a Multi-Port Memory Controller (MPMC). This controller provides access to RAM module for multiple peripherals and the Microblaze processor individually.

![Figure 4-3: Multi Port Memory Controller Module and Interface options for XUPV5](image)

The custom hardware component must provide the control signals with the correct timing to access the MPMC. The timing of control signals must be accurate in order to transmit the data correctly. Unlike software, the hardware has full control over how the data is to be transmitted. The MPMC is essentially a memory arbitrator which regulates
4. Design and Development

the access to the DDR2 RAM from multiple locations. It is often that a request from one port is put on hold while another is being processed. This is not unlike the design of a modern PC which also has a DDR interface system.

As illustrated in Figure 4-3, the MPMC supports different types of data buses connections to its ports. Table 4-1 summarizes the available data buses on the Microblaze system. The native port interface (NPI) is chosen as the communication bus between the custom hardware component and MPMC as it uses simplified signaling and is well documented.

<table>
<thead>
<tr>
<th>Bus Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AXI</td>
<td>Advanced eXtensible Interface</td>
</tr>
<tr>
<td>PLB</td>
<td>Processor Local Bus</td>
</tr>
<tr>
<td>NPI</td>
<td>Native Port Interface</td>
</tr>
<tr>
<td>XCL</td>
<td>Xilinx Cache Link</td>
</tr>
</tbody>
</table>

For the proposed design, the hardware component will be controlled by several sequential control registers which are mapped into the kernel address space. These registers will provide essential information to the hardware, such as which functions to perform as well as other key parameters (e.g. RAM locations, window size, colour, etc.). The hardware, when commanded, will access the physical DDR2 memory which has been set aside from the kernel. This memory will contain the bitmaps of the glyphs, their metrics, and the arrays which the hardware will generate. Since the hardware component
will have direct access the frame buffer, there will be no need to transfer the final surface back to the software.

### 4.3.2 Hardware-Functions

The hardware component supports three distinct functions: Glyph loading, Layout, and Placement.

Glyph loading loads the metrics (e.g. bitmap width, height, Y bearing, etc.) of the font data provided by FreeType into internal registers for faster access. Again, for testing, up to 128 possible glyphs are supported using a simple caching policy. It is up to the software to determine, if necessary, a better caching strategy. This information is translated to the hardware by use of the text array. This array is arbitrary as it only references the characters based on an index. This index can be altered to better suite a different policy, for example, index 1 may be the most referred to character, so the text array would have a 1 where that character is located.

The layout function creates the four arrays mentioned earlier as it processes the text array. The advance array is updated every character while the other are updated once a line is processed. This function will perform the kerning operation if enabled by the control registers.
The layout phase utilizes the information from the placement phase to copy the glyph bitmaps to the destination surface. It performs all the necessary address configurations and performs the copy.

All of the above operations output a “done” flag to the control registers so the software will be aware of when the task is completed. The software can execute other operations while these tasks are being performed.

Appendix A offers a guide for compiling the hardware and software designs, Appendix B contains the necessary system files to build the correct working design in XPS, and Appendix C contains the VHDL code of the designed hardware and development environment. Figure 4–4 illustrates the block diagram of the final system which houses the proposed hardware engine as one of the peripherals.
4. Design and Development

The figure shows two main buses in the design, the Microblaze Processor Local Bus (mb_plb) and the Native Port Interface (NPI). The Microblaze processor connects
4. Design and Development

directly to a Block RAM for its instruction and data caches, and it connects to the MPMC for DDR2 access. It also connects to numerous devices via the mb_plb which are necessary for the system to run Linux (timers, interrupt controllers, debuggers, etc.). The display hardware (XPS_TFT) connects as a master to the mb_plb bus as it needs guaranteed data access to update the extern display. The proposed hardware connects to the mb_plb bus in order to facilitate the control registers, and the NPI for DDR2 RAM access.

4.3.3 Application Programming Interface

Any hardware peripheral in a modern computer system today must be paired with a software driver in order to properly control the peripheral. Drivers are software running in the kernel which has access to the hardware control registers of the peripheral. Writing drivers is very specific to the kernel version and can cause system instability especially during development.

Fortunately and alternative approach is possible by the using the OS function “mmap” and “/dev/mem”. “/dev/mem” is a pseudo file to Linear Physical RAM on the system. When read, it will return the contents of RAM starting from its base address. “mmap” allows files to be mapped to memory addresses so any writes to RAM are actually committed to the file. In this case, “/dev/mem” is the file and direct RAM access is facilitated.
Communication with the hardware is done by access to the upper portions of DDR2 RAM for which the kernel has been told never to use, it is therefore safe. The proposed systems API’s responsibility is to prepare data for the hardware by copying it from virtual addresses (physically fragmented blocks of data) to this area in high RAM (linear blocks).

FreeType is used by the proposed design to render the individual glyphs, but it uses its own memory management system and does not allow a third party interface. This requires the API to copy the rendered glyphs from virtual RAM addresses to the upper communication space, which is an added bottleneck. This could be improved by modifying FreeType to allow for this feature, but is out of the scope of this work. Further analysis of this double memory overhead penalty is necessary in order to make a proper decision.

The API precedes the hardware functions by preparing the data for them. For example, prior to executing the “glyph loading” function noted in the previous section, the API prepares the memory interface (mmap), loads the glyphs, renders them, extracts the metrics, and transfers this information into the upper DDR2 RAM area. The hardware function is then enabled by accessing the control register. Once the hardware is done (indicated by changing the status bit on the control register), the API prepares any returned data by copying it back into the virtual address space of the process and cleans up (munmap - inverse of mmap). Similar coding is used for the layout and placement
functions. The proposed hardware is limited in the amount of information it can handle; limitations are necessary in hardware design. The API is responsible for breaking up a request into a sequence of operations. For example, a layout of a very large text passage may require hundreds of hardware layout functions. The API would set each up and sequence them accordingly and the calling function would none the wiser.

4.4 Summary

This chapter briefly covered the proposed design and implementation details. The reader is encouraged to examine the contents of the Appendix for a complete description of the design and the steps necessary to implement it.
Chapter 5

Results

5.1 Introduction

With the hardware design and software API complete, performance tests can be completed to determine the effects of the hardware acceleration in the text display process. All of the results are based on timing performance. Although a power analysis would be useful, it is difficult to perform within this environment as it requires isolation of the individual components of the development board.

This chapter shows the results of the raw hardware system compared to a streamlined software approach as well as comparisons with WebKit with and without the acceleration.

5.2 Raw Engine Performance

The raw engine is tested outside of WebKit to determine its performance with minimal software overhead. The software in this case simply reads an ASCII file and calls the API to layout (with no visual kerning) and places the contents on the frame buffer, which is displayed on the attached screen. Table 5-1 shows the timing result for different character sizes and lengths of text for the software only system.
5. Results

Table 5-1: Raw software timing results with no kerning

<table>
<thead>
<tr>
<th>Text Length (Characters)</th>
<th>Glyph Rasterizing (ms)</th>
<th>Layout Design (ms)</th>
<th>Bitmap Placement (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>14 24 32 48</td>
<td>14 24 32 48</td>
<td>14 24 32 48</td>
</tr>
<tr>
<td>1000</td>
<td>495 575 645 725</td>
<td>47 47 47 47</td>
<td>92 158 235 505</td>
</tr>
<tr>
<td>2000</td>
<td>495 575 645 725</td>
<td>101 101 101 101</td>
<td>195 340 500 1050</td>
</tr>
<tr>
<td>5000</td>
<td>495 575 645 725</td>
<td>227 227 227 227</td>
<td>445 770 1140 2410</td>
</tr>
<tr>
<td>10000</td>
<td>495 575 645 725</td>
<td>471 471 471 471</td>
<td>950 1645 2370 5004</td>
</tr>
<tr>
<td>100000</td>
<td>495 575 645 725</td>
<td>4,660 4,660 4,660 4,660</td>
<td>9,320 16,095 23,160 49,570</td>
</tr>
<tr>
<td>1000000</td>
<td>495 575 645 725</td>
<td>47,840 47,840 47,840 47,840</td>
<td>96,200 165,150 237,580 505,100</td>
</tr>
</tbody>
</table>

Table 5-2 shows the proposed software-hardware hybrid with the same parameters.

Table 5-2: Raw software hardware hybrid timing results with no kerning

<table>
<thead>
<tr>
<th>Text Length (Characters)</th>
<th>Glyph Rasterizing (ms)</th>
<th>Layout Design (ms)</th>
<th>Bitmap Placement (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>14 24 32 48</td>
<td>14 24 32 48</td>
<td>14 24 32 48</td>
</tr>
<tr>
<td>1000</td>
<td>495 575 645 725</td>
<td>0.8 0.8 0.8 0.8</td>
<td>9.6 22 36 82</td>
</tr>
<tr>
<td>2000</td>
<td>495 575 645 725</td>
<td>1 1 1 1</td>
<td>20 47 78 172</td>
</tr>
<tr>
<td>5000</td>
<td>495 575 645 725</td>
<td>3 3 3 3</td>
<td>47 104 170 382</td>
</tr>
<tr>
<td>10000</td>
<td>495 575 645 725</td>
<td>6 6 6 6</td>
<td>94 212 350 790</td>
</tr>
<tr>
<td>100000</td>
<td>495 575 645 725</td>
<td>61 63 64 68</td>
<td>884 2,050 3,358 7,650</td>
</tr>
<tr>
<td>1000000</td>
<td>495 575 645 725</td>
<td>610 630 640 660</td>
<td>9,060 21,080 34,454 78,550</td>
</tr>
</tbody>
</table>

These results show a substantial improvement in layout performance (average 77X faster) and placement (average 7.6X faster). Placement performance doesn’t scale as well as layout due to memory access speeds. This is a design flaw in the algorithm due to the way DDR2 RAM is accessed. In this case, memory access only reads and writes a single word in one request. This can be improved by using Burst memory access (discussed later in this chapter).
5. Results

As visual kerning only affects the layout portion of the proposed engine, Table 5-3 shows the impact it has on timing.

<table>
<thead>
<tr>
<th>Font Size (Pixels)</th>
<th>Layout Design with Visual Kerning (ms)</th>
<th>Layout Design without Visual Kerning (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5,715</td>
<td>1,335</td>
</tr>
<tr>
<td>14</td>
<td>7,059</td>
<td>1,340</td>
</tr>
<tr>
<td>18</td>
<td>8,521</td>
<td>1,344</td>
</tr>
<tr>
<td>24</td>
<td>10,900</td>
<td>1,353</td>
</tr>
<tr>
<td>32</td>
<td>13,886</td>
<td>1,370</td>
</tr>
<tr>
<td>48</td>
<td>20,923</td>
<td>1,383</td>
</tr>
<tr>
<td>56</td>
<td>23,883</td>
<td>1,394</td>
</tr>
</tbody>
</table>

It can be seen that visual kerning requires more and more time as the characters height increases. This is because each row of each glyph has to be compared with its neighbor thus requiring more cycles of operation.

5.3 WebKit Engine Performance

Since WebKit is the underlying web rendering engine used in most mobile browsers, it makes sense to compare its stock performance (software only) with the proposed engine. To do this, WebKit was modified to use the custom API to enable use of the hardware acceleration.
5. Results

Table 5-4 shows the result of performance results between the stock WebKit rendering engine, and the accelerated WebKit with and without visual kerning for a 2 million character text passage.

Table 5-4: Performance comparison between proposed engine and WebKit in Layout Design

<table>
<thead>
<tr>
<th>Font Size (Pixels)</th>
<th>Stock WebKit (ms)</th>
<th>Proposed Engine with visual kerning (ms)</th>
<th>Proposed Engine without visual kerning (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>167,960</td>
<td>5,715</td>
<td>1,335</td>
</tr>
<tr>
<td>14</td>
<td>244,120</td>
<td>7,059</td>
<td>1,340</td>
</tr>
<tr>
<td>18</td>
<td>270,270</td>
<td>8,521</td>
<td>1,344</td>
</tr>
<tr>
<td>24</td>
<td>300,816</td>
<td>10,900</td>
<td>1,353</td>
</tr>
<tr>
<td>32</td>
<td>315,235</td>
<td>13,886</td>
<td>1,370</td>
</tr>
<tr>
<td>48</td>
<td>(Crashed)</td>
<td>20,923</td>
<td>1,383</td>
</tr>
<tr>
<td>56</td>
<td>(Crashed)</td>
<td>23,883</td>
<td>1,394</td>
</tr>
</tbody>
</table>

These results show that even when considering the heavy computation time of visual kerning, the proposed engine performs the layout on average 29 times faster than the software rendering engine of WebKit. Without visual kerning the proposed engine can perform on average 192 times faster than WebKit; as the text passage gets larger, the faster the overall performance. It is interesting to note how the proposed engine’s timing is essentially constant without kerning, but WebKit’s isn’t. This is likely due to the fact of the software and memory management overhead. WebKit is designed for maintainability not necessarily performance. Also note that WebKit crashed at larger font sizes, probably due to memory restrictions.
Assessing the performance of placement in WebKit is far more difficult than originally considered. As mentioned before, WebKit computes its surface bitmap based on a predictive nature. It was not possible to force WebKit to perform long placement sequences as it would optimize out the actions. For example, asking WebKit to “page down” 5 times resulted in it showing the first page and page 6, 2 to 5 were skipped entirely. Some results can be shown, but they are of small text passages, the size of this predictive cache. Table 5-5 shows these results.

Table 5-5: A timing comparison for Bitmap Placement Phase between WebKit and proposed engine

<table>
<thead>
<tr>
<th>Text Length (Characters)</th>
<th>WebKit (ms)</th>
<th>Proposed Engine (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>420</td>
<td>80</td>
</tr>
<tr>
<td>2000</td>
<td>900</td>
<td>170</td>
</tr>
<tr>
<td>5000</td>
<td>1,540</td>
<td>330</td>
</tr>
<tr>
<td>8000</td>
<td>2,500</td>
<td>500</td>
</tr>
<tr>
<td>16000</td>
<td>5,100</td>
<td>1030</td>
</tr>
</tbody>
</table>

The improvement in this phase is constant at about 5 times faster. This is the time it takes for WebKit to strictly perform the placement on the surface. Additional time is necessary to transfer the surface to the frame buffer.

5.4 Effect of Burst Memory Access

As noted earlier in chapter 4, the MPMC is an arbiter for DDR2 memory access in the Microblaze system. Any modern DDR memory (i.e. DDR, DDR2, DDR3, etc.) has a
significant latency to retrieve and store data. The effect of this latency is reduced by higher clock speeds. In the case of the proposed design, for the sake of design simplicity, all memory access is single-word only; that is the MPMC is asked to only retrieve and store single 32-bit words during each transaction. The latency measured on the current system is 23 cycles; this is typical of DDR2 memory. Unfortunately it slows down DDR2 access with small word requests.

It is possible to improve the design to allow for “burst” reads and writes in groups of 2, 4, 8, 16 or 32 words. When operating in groups, the remaining words are received or transmitted on each clock cycle after the first is received or sent [33]. The improvement in latency is significant as illustrated in the example below.

• Read an 18x18 Character:
  o With using only Single Word Read:
    \[ 18 \times (18 \times (\text{Latency-Read Word})) = 18 \times 18 \times 23 = 7452 \text{ cycles} \]
  o Using 16 Word Burst Read and 2 Word Burst Reads:
    \[ 18 \times (\text{Latency-Burst Read 16 Words} + \text{Latency-Burst Read 2 Words}) = 18 \times (23 + 15 + (23 + 1)) = 18 \times 62 = 1116 \text{ cycles} \]

• Read a 32 x 32 Character:
  o With using only Single Word Read:
    \[ 32 \times (32 \times (\text{Latency-Read Word})) = 32 \times 32 \times 23 = 23552 \text{ cycles} \]
  o Using 32 Word Burst Read:
    \[ 32 \times (\text{Latency-Burst Read 32 Words}) = 32 \times (23 + 31) = 32 \times 54 = 1728 \text{ cycles} \]
Although the proposed engine does not use burst read access, we can approximate the general improvement by statistically analyzing the averages cycles necessary to read 32 words without burst.

\[
\text{AvgCycNoBurst} = \frac{\sum_{n=1}^{32} 23 \times n}{32} = 379.5
\]

When using burst, a choice can be made whether or not to break up the required word reads into their individual components or to simple read more data to save clock cycles. For example, reading 31 words with burst would be a 16, 8, 4, 2, and 1 word read; the total cycles would be \((23+15)+(23+7)+(23+3)+(23+1)+(23)=141\). The average cycles for 32 reads would be \(72.1875\) (calculated with spread sheet). However if one simply reads 32 words (and discards the last value), the cycles would be \((23+31) = 54\) which requires fewer cycles. The average cycles while reading more words than necessary is \(43.34375\) (calculated with spread sheet), which is fewer cycles than decomposing the read into its minimal components. Based on this, we could say that by using burst read, we can improve the read performance by \(379.5/43.34375\) or 8.75 times. Unfortunately we cannot do the same for the write operation as we cannot over write the specified boundaries. In this case, the write operation would have to be decomposed, so the average speedup would be \(379.5/72.1875\) or 5.25 times.

In terms of an overall improvement factor, since the placement phase is essentially a memory copy, we can say one half of the cycles are reading, and the other are writing.
The lump improvement is therefore $8.75/2 + 5.25/2$ or 7 times. Therefore it is reasonable to assume that the best possible speed improvement with burst operations in the placement phase will be 7 times that without burst. In order to achieve this rate, the design would have to be changed to ensure the read and write operations are executed back-to-back with minimal delay.

It is important to note that the Microblaze CPU utilizes both and instruction and data cache system so it is performing burst reads and writes whenever possible. This is contributing to its performance.

### 5.6 Hardware Resources Required

As discussed in previous sections, the designed hardware-software engine is implemented using a Xilinx FPGA board. Table 5-6 shows the amount of resources used by the designed custom hardware. Two scenarios are presented here, one where registers are used for font metric storage, the other (an approximate) if block RAMs are used. This latter scenario has not been implemented as of yet, but this shows the proposed design is dominated by these storage registers.

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Using Registers</th>
<th>Using Block RAMs (Approximate)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice Registers</td>
<td>5100</td>
<td>2029</td>
</tr>
<tr>
<td>Slice LUTs</td>
<td>5681</td>
<td>3426</td>
</tr>
<tr>
<td>LUT FF Pairs used</td>
<td>9620</td>
<td>4284</td>
</tr>
<tr>
<td>Fully used LUT FF Pairs</td>
<td>1161</td>
<td>1171</td>
</tr>
<tr>
<td>Unique Control Sets</td>
<td>291</td>
<td>99</td>
</tr>
</tbody>
</table>
5. Results

5.5 Power Savings

Although there has been no formal evaluation of power, considering the savings in clock cycles alone using the proposed engine, it is reasonable to assume that there will be a good chance for some power savings. Power is a measure of energy used over time. When the number of cycles to perform the operation is reduced significantly, the time it takes to execute the operation is also reduced; this in turn will reduce power. The question is, how much of a reduction. To answer this question with confidence in finite time, very sensitive current measuring equipment is required which can differentiate power consumption at very small levels between all components of the Xilinx development board and the individual components inside the FPGA. This is very difficult to do as the development board used does not offer any component isolation. Another alternative is to use a software approach where the switching is determined by running the whole system in a simulation and measuring the switching activity. Since the simulations results shown here require tens of minutes of real operating time, this translates into months of simulation time. Also, all the hardware components of the development board are not modeled so a proper simulation is not possible.
5. Results

5.6 Summary

This chapter presented the final timing results of the proposed engine. They show that under raw conditions the hardware engine can accelerate layout, on average, 77 times (based on the content length). Placement times are increased on average 7.6 times, this is slower than the layout step but with full use of the memory controller, a maximum of 53 times performance improvement (7X more) is expected.

When used in WebKit, layout times are, on average, 29 and 192 times faster with and without kerning respectively. This significant difference compared to the raw performance above is likely due to the methods in which WebKit is coded, for organization and maintenance, not necessarily performance. This result also shows that it is possible to improve the glyph spacing using kerning with no significant penalty on performance while using this hardware. Although placement is more difficult to measure in WebKit, we have showed a 5 times improvement (with short text lengths), but with only single-word memory access to the DDR2 controller. Should the hardware system be improved to fully utilize burst modes, this improvement could be at most 35 times faster.

Although improvements in regards to power were not formally performed, given the reducing in clock cycles to complete the display tasks, there is a good chance there has been some savings.

Figures 5-1 below shows graphically the potential improvements made.
5. Results

(a) Software Rendering Engine

(b) Proposed Engine without Burst Access

(c) Proposed Engine with Burst Access

Figure 5-1: Effect of Burst Access
Chapter 6

Conclusion

6.1 Conclusion

The intent of this work was to determine if the proportional text display process used in today’s GUIs could be accelerated by hardware to improve the general user experience on mobile devices. Through analysis on similar hardware it was found that two key steps could benefit from some type of acceleration: layout, the planning of the individual character spacing, and placement, the process which the characters are placed on an image bitmap. In order to execute a fair comparison between both software and hardware accelerated implementations, an FPGA based embedded system was selected which utilizes a soft-core CPU. Although the performance of the CPU is not in the same league as modern mobile CPUs, the value of the comparison is important as the environment between the contenders is identical.

The timing results showed that with the proposed hardware, under raw conditions, improved the layout by 77 times and placement by 7.6 times on average. The effect of kerning on performance varies with font size, but is easily 5 times faster even with small text lengths. When applied in WebKit, the layout is shown to be 29 and 192 times faster with and without kerning respectively, while placement is about 5 times faster. The
placement performance, in both cases, is hindered by the method of which memory is accessed on the embedded system. It was statistically shown that an additional 7 times performance could be had if burst memory accesses were performed.

Although power performance was not measured, the substantial improvement in both of these key areas indicates there is a good chance for power savings as the number of clock cycles has been reduced greatly.

**6.2 Contributions**

This work has generated a number of contributions. First and foremost the FPGA evaluation environment is crucial to determining the performance differences between software and hardware implementations while remaining in the same environment. The appendices detail the necessary steps to build the system from the beginning to a fully functional system.

The hardware itself is based on a state machine design which allows easy modification to add new features. Although including burst operations is challenging, it would not require a full design. Other future hardware accelerations, perhaps image based algorithms, can be easily added.

This work also introduces a new kerning algorithm known as “visual kerning”. The algorithm calculates the space between the glyph and bounding box for each row to
6. Conclusion

determine the best fit with neighboring glyphs. When included in the hardware layout system, visual kerning out performs standard software placement.

6.3 Recommendations for Future Work

As with any other hardware design, it is always possible to further optimize it. Any changes to the internal state machine could yield further improvements.

The placement phase could be substantially improved, as shown in chapter 5, by use of the burst memory access. This work would be challenging as an intermediate buffer would be required (a Block RAM is suggested), but it certainly doesn’t require a complete redesign.

Now that the Xilinx Zynq platform is available, the design should be ported to it. Since the Zynq system is similar to the Microblaze system, this porting shouldn’t be terribly difficult.

The presented work utilized a simple glyph caching policy, that is cache everything. Existing methods should be researched and implemented to improve the API functionality.

The placement phase in WebKit is largely fragmented which makes the integration with the proposed system’s API difficult. Further work should be done here, perhaps a modification to WebKit, to allow a simpler fit.
References


References


[26] “Xilinx university program XUPV5-LX110T development system”, , January 2011,


References


[33] “LogiCORE IP Multi-Port Memory Controller MPMC v6.03.a”, Xilinx Inc., March 2011,

[34] “Petalinux SDK Board Bring up Guide”, Petalogix Inc., March 2011,

Appendix A

System Design Procedure

This appendix will go through the procedure of designing hardware and software elements of the system in details. It will serve for future colleagues and the reader as a design manual, if one desires to further improve the developed engine.

A.1 Base System Builder

In order to build the discussed system of this work from scratch, the developer will need the following tools:

- Xilinx XUPV5-LX110T FPGA Development Kit,
- Xilinx Design Studio (XPS 12.4 tool is used in this work),
- Petalinux v2.1 Final,

The first step is to build the base system. It can be done by using the Base System Builder setup in XPS. In the Base System Builder the developer must choose PLB system as the type of the embedded system design. This will generate base file for a system based on a PLB System BUS.

In the next menu the Board configurations must be chosen as follows:

Board Vendor: Xilinx
Board Name: Virtex5 ML505 Evaluation Platform
Board Revision: 1
The rest of the options should remain unchanged. A Single Processor System must be chosen from the next menu as it is needed for this work to have single soft core Microblaze processor to control the system in whole.

The Processor Configuration in the next menu should be as follows:

Processor Type: Microblaze  
System Clock Frequency: 125.00 MHz  
Local Memory: 8 KB  

There are several options for the clock speed of the system. The highest for this board 125 MHz and lower ones can also be used. The author has developed two versions of the system one with 125 MHz clock frequency and another one with 75 MHz frequency.

The final design with 125 MHz clock frequency for the system has some untreated timing violations and might result into a non-stable system. The 75 MHz version will not have any timing violations. Test results of this work are taken from both builds and compared with each other.

There is no need to enable the floating point unit for the system, since it will not be used in any special task for the special design.

Xilinx Design Studio has many different peripherals available in its library to add to the system. In order to have a simpler and more reliable system, only ones that are needed for the work will be added to the base system. Figure A-1 shows the required peripherals and their parameters as is shown in XPS 12.4.
The Instruction and Data Cache must both be enabled in the next menu. The size of both should be changed to 4 KB and DDR2_SDRAM must be chosen as the Cache Memory for both.

There is no need to apply any changes for the Application menu. There will be summary of the system be shown at the last step and the base system build is finished at this point.
A. System Design Procedure

A.2 Convert ML505 Design to XUPV5

The board type chosen in the previous section for the base system builder wizard was ML505. This is due to not having the option of choosing XUPV5 as the board type. ML505 and XUPV5 are same FPGA boards in principle and both use Virtex5 technology. There are only some minor differences in the pin layouts and the size of the FPGA.

After the base system builder wizard is done, the target device must be changed. In the project options menu in XPS the device size must be changed to xc5vlx110t. The package must be ff1136 and the speed gate must show -1.

In order for the design to work there are few lines that must be changed in the system.ucf file. The following lines must be deleted from the system.ucf file:

```
# LOC placement of DQS-squelch related IDDR and IDELAY elements
# Each circuit can be located at any of the following locations:
# 1. Unused "N"-side of DQS diff pair I/O
# 2. DM data mask (output only, input side is free for use)
# 3. Any output-only site

INST "/gen_dqs[0].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y56";
INST "/gen_dqs[1].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y18";
INST "/gen_dqs[2].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y60";
INST "/gen_dqs[3].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y62";
INST "/gen_dqs[4].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y216";
INST "/gen_dqs[5].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y220";
INST "/gen_dqs[6].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y222";
INST "/gen_dqs[7].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y224";
```

A. System Design Procedure

# LOC and timing constraints for flop driving DQS CE enable signal
# from fabric logic. Even though the absolute delay on this path is
# calibrated out (when synchronizing this output to DQS), the delay
# should still be kept as low as possible to reduce post-calibration
# voltage/temp variations - these are roughly proportional to the
# absolute delay of the path

INST "/u_phy_calib_0/gen_gate[0].u_en_dqs_ff" LOC = SLICE_X0Y28;
INST "/u_phy_calib_0/gen_gate[1].u_en_dqs_ff" LOC = SLICE_X0Y9;
INST "/u_phy_calib_0/gen_gate[2].u_en_dqs_ff" LOC = SLICE_X0Y11;
INST "/u_phy_calib_0/gen_gate[3].u_en_dqs_ff" LOC = SLICE_X0Y30;
INST "/u_phy_calib_0/gen_gate[4].u_en_dqs_ff" LOC = SLICE_X0Y31;
INST "/u_phy_calib_0/gen_gate[5].u_en_dqs_ff" LOC = SLICE_X0Y108;
INST "/u_phy_calib_0/gen_gate[6].u_en_dqs_ff" LOC = SLICE_X0Y110;
INST "/u_phy_calib_0/gen_gate[7].u_en_dqs_ff" LOC = SLICE_X0Y111;

These lines represent the DDR2 constraint. After removing the above line, the
following lines must be added to the UCF file to replace the removed DDR2 constraint
with new one:

INST "/gen_dqs[0].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y96";
INST "/gen_dqs[0].u_iob_dqs/u_iodelay_dq_ce" LOC = "IODELAY_X0Y96";
INST "/gen_dqs[1].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y58";
INST "/gen_dqs[1].u_iob_dqs/u_iodelay_dq_ce" LOC = "IODELAY_X0Y58";
INST "/gen_dqs[2].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y62";
INST "/gen_dqs[2].u_iob_dqs/u_iodelay_dq_ce" LOC = "IODELAY_X0Y62";
INST "/gen_dqs[3].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y100";
INST "/gen_dqs[3].u_iob_dqs/u_iodelay_dq_ce" LOC = "IODELAY_X0Y100";
INST "/gen_dqs[4].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y102";
INST "/gen_dqs[4].u_iob_dqs/u_iodelay_dq_ce" LOC = "IODELAY_X0Y102";
INST "/gen_dqs[5].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y256";
INST "/gen_dqs[5].u_iob_dqs/u_iodelay_dq_ce" LOC = "IODELAY_X0Y256";
INST "/gen_dqs[6].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y260";
INST "/gen_dqs[6].u_iob_dqs/u_iodelay_dq_ce" LOC = "IODELAY_X0Y260";
INST "/gen_dqs[7].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y262";
INST "/gen_dqs[7].u_iob_dqs/u_iodelay_dq_ce" LOC = "IODELAY_X0Y262";

INST
"DDR2_SDRAM/DDR2_SDRAM/gen_no_iodelay_grp.genInstantiate_idelayctrls[1].idelayctrl0" LOC = IDELAYCTRL_X0Y2;
INST
"DDR2_SDRAM/DDR2_SDRAM/gen_no_iodelay_grp.genInstantiate_idelayctrls[0].idelayctrl0" LOC = IDELAYCTRL_X0Y6;
INST
"DDR2_SDRAM/DDR2_SDRAM/gen_no_iodelay_grp.genInstantiate_idelayctrls[2].idelayctrl0" LOC = IDELAYCTRL_X0Y1;
# LOC and timing constraints for flop driving DQS CE enable signal
# from fabric logic. Even though the absolute delay on this path is
calibrated out (when synchronizing this output to DQS), the delay
should still be kept as low as possible to reduce post-calibration
voltage/temp variations - these are roughly proportional to the
absolute delay of the path

INST "/u_phy_calib_0/gen_gate[0].u_en_dqs_ff" LOC = SLICE_X0Y48;
INST "/u_phy_calib_0/gen_gate[1].u_en_dqs_ff" LOC = SLICE_X0Y29;
INST "/u_phy_calib_0/gen_gate[2].u_en_dqs_ff" LOC = SLICE_X0Y31;
INST "/u_phy_calib_0/gen_gate[3].u_en_dqs_ff" LOC = SLICE_X0Y50;
INST "/u_phy_calib_0/gen_gate[4].u_en_dqs_ff" LOC = SLICE_X0Y51;
INST "/u_phy_calib_0/gen_gate[5].u_en_dqs_ff" LOC = SLICE_X0Y128;
INST "/u_phy_calib_0/gen_gate[6].u_en_dqs_ff" LOC = SLICE_X0Y130;
INST "/u_phy_calib_0/gen_gate[7].u_en_dqs_ff" LOC = SLICE_X0Y131;

One more change must be made to the design before compiling the base system. In the
project option menu, from the Design Flow tab, the option to treat timing closure failure
as an error must be disabled.

At this point, the base system is ready for Bitstream generation. The system is not
complete yet, but in order to test the design, it is better to run Generate Bitstream from
the Hardware menu and make sure the design compiles fine up to this step.

**A.3 TFT Display Controller**

Having a display controller is not necessary for an embedded pc to work but for this
special project in order to visually see the output result a display controller is required to
be added to the system. There is a display controller IP Core available in the Xilinx EDK
library named XPS_TFT_2.01_a. Adding the TFT controller IP Core should be a straight
forward task like other peripherals, but for some reason there are few more steps to be
A. System Design Procedure

taken more than the standard procedure in order to have a working TFT Controller IP
Core in the system.

The first step is to add the XPS_TFT from IP Catalog menu in XPS. The XPS_TFT IP
Core can be found under subsection IO Modules. Next the data buses of the peripheral
must be connected to the PLB BUS. XPS_TFT has two buses named MPLB and SPLB
which represent Master PLB and Slave PLB. Both must be connected to mb_plb bus
from the Bus Interface menu.

At this point there are few lines that are needed to be added to the end of the system.ucf
file. The following lines will declare the pin connections for the XPS_TFT and must be
manually added to the system.ucf file.

```asciidoc
# IO Pad Location Constraints / Properties for TFT LCD Controller

NET xps_tft_0_TFT_IIC_SCL  LOC = U27;
NET xps_tft_0_TFT_IIC_SDA  LOC = T29;
NET xps_tft_0_TFT_IIC_SCL  SLEW = SLOW;
NET xps_tft_0_TFT_IIC_SCL  DRIVE = 6;
NET xps_tft_0_TFT_IIC_SCL  TIG;
NET xps_tft_0_TFT_IIC_SCL  IOSTANDARD = LVCMOS18 ; #ff LVCMOS33;
NET xps_tft_0_TFT_IIC_SDA  SLEW = SLOW;
NET xps_tft_0_TFT_IIC_SDA  DRIVE = 6;
NET xps_tft_0_TFT_IIC_SDA  TIG;
NET xps_tft_0_TFT_IIC_SDA  IOSTANDARD = LVCMOS18 ; #ff LVCMOS33;
NET xps_tft_0_TFT_DVI_DATA_pin<0> LOC = AB8;
NET xps_tft_0_TFT_DVI_DATA_pin<1> LOC = AC8;
NET xps_tft_0_TFT_DVI_DATA_pin<2> LOC = AN12;
NET xps_tft_0_TFT_DVI_DATA_pin<3> LOC = AP12;
NET xps_tft_0_TFT_DVI_DATA_pin<4> LOC = AA9;
NET xps_tft_0_TFT_DVI_DATA_pin<5> LOC = AA8;
NET xps_tft_0_TFT_DVI_DATA_pin<6> LOC = AM13;
NET xps_tft_0_TFT_DVI_DATA_pin<7> LOC = AN13;
NET xps_tft_0_TFT_DVI_DATA_pin<8> LOC = AA10;
NET xps_tft_0_TFT_DVI_DATA_pin<9> LOC = AB10;
NET xps_tft_0_TFT_DVI_DATA_pin<10> LOC = AP14;
NET xps_tft_0_TFT_DVI_DATA_pin<11> LOC = AN14;
NET xps_tft_0_TFT_DVI_DATA_pin<*> IOSTANDARD = LVDCI_33;
NET xps_tft_0_TFT_DVI_CLK_P_pin LOC = AL11;
```
A. System Design Procedure

NET xps_tft_0_TFT_DVI_CLK_P_pin IOSTANDARD = LVCMOS33 | DRIVE = 24 | SLEW = FAST;
NET xps_tft_0_TFT_DVI_CLK_N_pin LOC = AL10;
NET xps_tft_0_TFT_DVI_CLK_N_pin IOSTANDARD = LVCMOS33 | DRIVE = 24 | SLEW = FAST;
NET xps_tft_0_TFT_HSYNC_pin LOC = AM12;
NET xps_tft_0_TFT_HSYNC_pin IOSTANDARD = LVDCI_33;
NET xps_tft_0_TFT_VSYNC_pin LOC = AM11;
NET xps_tft_0_TFT_VSYNC_pin IOSTANDARD = LVDCI_33;
NET xps_tft_0_TFT_DE_pin LOC = AE8;
NET xps_tft_0_TFT_DE_pin IOSTANDARD = LVDCI_33;
NET xps_tft_0_reset_pin LOC = AK6;
NET xps_tft_0_reset_pin IOSTANDARD = LVCMOS33;

Also the developer must make sure that the definition of the XPS_TFT in the system.mhs file looks exactly as the following:

BEGIN xps_tft
PARAMETER INSTANCE = xps_tft_0
PARAMETER HW_VER = 2.01.a
PARAMETER C_DCR_SPLB_SLAVE_IF = 1
PARAMETER C_SPLB_BASEADDR = 0x86e00000
PARAMETER C_SPLB_HIGHLADDR = 0x86e0ffff
PARAMETER C_TFT_INTERFACE = 1
PARAMETER C_I2C_SLAVE_ADDR = 0b1110110
PARAMETER C_DEFAULT_TFT_BASE_ADDR = 0x90000000
BUS_INTERFACE MPLB = mb_plb
BUS_INTERFACE SPLB = mb_plb
PORT TFT_HSYNC = xps_tft_0_TFT_HSYNC
PORT TFT_VSYNC = xps_tft_0_TFT_VSYNC
PORT TFT_DE = xps_tft_0_TFT_DE
PORT TFT_DVI_CLK_P = xps_tft_0_TFT_DVI_CLK_P
PORT TFT_DVI_CLK_N = xps_tft_0_TFT_DVI_CLK_N
PORT TFT_DVI_DATA = xps_tft_0_TFT_DVI_DATA
PORT TFT_IIC_SCL = xps_tft_0_TFT_IIC_SCL
PORT TFT_IIC_SDA = xps_tft_0_TFT_IIC_SDA
PORT IP2INTC_Irpt = xps_tft_0_IP2INTC_Irpt
PORT SYS_TFT_Clk = clk_25_0000MHz
END

There are few more changes that are needed to be made to the system.mhs file. There are some ports that are needed to be made as external. This will be done by adding following lines to the first section of the system.mhs file where the ports are defined.

PORT xps_tft_0_TFT_HSYNC_pin = xps_tft_0_TFT_HSYNC, DIR = O
PORT xps_tft_0_TFT_VSYNC_pin = xps_tft_0_TFT_VSYNC, DIR = O
PORT xps_tft_0_TFT_DE_pin = xps_tft_0_TFT_DE, DIR = O
PORT xps_tft_0_TFT_DVI_CLK_P_pin = xps_tft_0_TFT_DVI_CLK_P, DIR = O
PORT xps_tft_0_TFT_DVI_CLK_N_pin = xps_tft_0_TFT_DVI_CLK_N, DIR = O
PORT xps_tft_0_TFT_DVI_DATA_pin = xps_tft_0_TFT_DVI_DATA, DIR = O, VEC = [11:0]
PORT xps_tft_0_TFT_IIC_SCL_pin = xps_tft_0_TFT_IIC_SCL, DIR = IO
PORT xps_tft_0_TFT_IIC_SDA_pin = xps_tft_0_TFT_IIC_SDA, DIR = IO
PORT xps_tft_0_reset_pin = sys_periph_reset_n, DIR = O
A. System Design Procedure

PORT xps_tft_0_IP2INTC_Irpt_pin = xps_tft_0_IP2INTC_Irpt, DIR = 0, SIGIS = INTERRUPT, SENSITIVITY = EDGE_RISING

Also as seen in the above declaration of XPS_TFT, it will use a 25MHz clock. This clock frequency does not exist in the system.mhs file by default since the automatic system builder won’t use it. In order to make this clock, the following lines must be added to the clock_generator section in the system.mhs file.

PARAMETER C_CLKOUT4_FREQ = 25000000
PORT CLKOUT4 = clk_25_0000MHz

There is one more element needed to be added to the system in order for the XPS_TFT to work. An IP Core name util_vector_logic must be added to the system. This IP Core is a way of implementing a logic function to be applied to a signal in the system. Because of the properties of the XPS_TFT the reset signal of the system must be changed. By adding the following lines to the system.mhs file the needed IP Core will be added to the system.

BEGIN util_vector_logic
PARAMETER INSTANCE = util_vector_logic_0
PARAMETER HW_VER = 1.00.a
PARAMETER C_OPERATION = not
PARAMETER C_SIZE = 1
PORT Op1 = sys_periph_reset
PORT Res = sys_periph_reset_n
END

After reloading the project in XPS the added IP Core will show up in the system. There is no need to change anything and this is the easy way to add this IP Core manually to the system.

When all the needed parts have been added to the system, the interrupt controller must be programmed in order to support the XPS_TFT as well. One must make sure that the declaration of the interrupt controller looks like the following in the system.mhs file.
A. System Design Procedure

BEGIN xps_intc
PARAMETER INSTANCE = xps_intc_0
PARAMETER HW_VER = 2.01.a
PARAMETER C_BASEADDR = 0x81800000
PARAMETER C_HIGHADDR = 0x8180ffff
BUS_INTERFACE SPLB = mb_plb
PORT Intr = RS232_Uart_1_Interrupt & Ethernet_MAC_IP2INTC_Irpt & xps_timer_0_Interrupt & fpga_0_Ethernet_MAC_MDINT_pin & xps_tft_0_IP2INTC_Irpt
PORT Irq = microblaze_0_Interrupt
END

A.4 Add Custom IP Core

At this the base system is ready and the custom-built IP Core of this work which is a Text Layout and Display Engine can be added to the system.

From the Hardware menu in XPS, one must choose Create or Import Peripheral to start adding the IP Core to the system. At this point XPS will ask to either create or import a peripheral. The correct choice is to Create templates for a new peripheral. In the next menu the repository for the new peripheral must be chosen. The safe choice is to choose to add the new IP Core to the XPS project as a local pcore and not to add it to the EDK repository.

After choosing a name and Version for the peripheral XPS will ask which kind of BUS will the peripheral use. The choice for this work is PLB v4.6 or Processor Local Bus. After this point, make sure that none of the options are chosen except for User logic software register in the next menu. There is no action for the next menu.

Now XPS will ask about how many software accessible registers are needed. For this work the number is 20.
A. System Design Procedure

There is no need to change anything in any menu after this point. By clicking next for the rest of menus we go forward to the last step. At the end the wizard will show a summary and pressing finish will create the template for a new peripheral.

XPS creates two files as follows:

\begin{verbatim}
{$Project Directory}/pcores/New_Peripheral_Name/hdl/vhdl/New_Peripheral_Name.vhd

{$Project Directory}/pcores/New_Peripheral_Name/hdl/vhdl/user_logic.vhd
\end{verbatim}

XPS will also create some other files but there is no need to modify them.

The New_Peripheral_Name.vhd contains port definitions and properties in order to have access to PLB bus and software accessible registers. On the other hand user_logic.vhd file is where the design will be placed. These files must be changed according to the codes in Appendix C.

After modifying VHDL files to the desired design, now the peripheral can be added to the system. From the Project Local Pcores sub section in the IP Catalog the designed IP can be added to the system. After adding the IP Core the data buses must be connected.

As mentioned in Chapter 4 of the dissertation, the IP Core of this work uses NPI BUS to communicate with the DDR2 RAM. In order to make this connection possible, some changes must be made to the MPMC or DDR2_SDRAM module in the system. By right clicking on the instance in the Bus Interface menu and choosing Configure IP, a new window will appear where the properties of the DDR2_SDRAM can be modified. Figure
A. System Design Procedure

A-2 to A-6 shows how the menus should look like after the necessary changes are made to the MPMC module. The memory interface is the crucial part in adding the new peripheral and NPI BUS interface so the author decided to show the configuration menus for the MPMC.

Figure A-2: How to configure MPMC Step 1
A. System Design Procedure

Figure A-3: How to configure MPMC Step 2

Figure A-4: How to configure MPMC Step 3
A. System Design Procedure

Figure A-5: How to configure MPMC Step 4

Figure A-6: How to configure MPMC Step 5
A. System Design Procedure

After properly configuring the MPMC module, the bus connections must be done for the new peripheral. There is no need to worry about the bus connections of the rest of the system, since the automated base system builder has already implemented all the connections. The Bus Interface for the new peripheral (The author’s design is named displaymem2_0) and the DDR2_SDRAM should look like the following figure at the end.

![Figure A-7: How to connect ports of the developed peripheral](image)

94
A. System Design Procedure

At this point there is only one step left before the hardware design is finished. The IO Mapped Addresses of the IPs of the system must be verified and the end result should look like the Figure A-8.

![Figure A-8: IO Mapped Memory Addresses of the system](image)

Now that the hardware is finished, by choosing the Generate Bitstream from the Hardware menu, the final system must be compiled. If everything goes fine, one can proceed to the next step to configure the Software Platform Settings of the system.

A.5 Software Platform Settings

It is required to indicate what Operating System the designed embedded pc will use. This can be done through the Software Platform Settings option in Software menu of XPS. In the OS & Library Settings of the menu petalinux must be chosen as the Operating System for this work.
In the OS and Lib Configuration tab, the configuration for OS must be modified as what is shown in figure A-9.

![Figure A-9: How to configure Petalinux](image)

The rest of the steps to configure the operating system and add fs-boot to the hardware as the boot loader are standard processes and can be found in [34].

The only point to consider is to add the following line in the Paths and Options tab of Compiler Options. The procedure is mentioned in [34] but the accurate command for the system designed in this work is the following:

```
```
A. System Design Procedure

-DCONFIG_FS_BOOT_OFFSET=0x200000 –Zxl-mode-novectors

When everything is ready, the hardware image can be downloaded to the FPGA by using Download Bitstream from the Device Configuration menu of XPS.

A.6 Compiling Petalinux Image

The next step after compiling the hardware image of the system is to compile the operating system based on the properties of the hardware. There are few options available to be used as the operating system for an embedded pc designed on a Xilinx FPGA.

Due to the nature of this work and text scenarios, a version of Linux is chosen to be used as the operating system. Petalinux from Petalogix [18] is a version of Linux intended to be used on an embedded pc. In order to use petalinux few configuration steps must be performed to come up with an OS image matching the properties of the hardware image of the system.

After successfully building the hardware image, the next step is to create template files for the OS image. By executing the following command a template directory with necessary files will be created.

$ petalinux-new-platform –c {CPU Architecture} –v {Vendor} –p {Platform}
A. System Design Procedure

{CPU Architecture} must be replaced by Microblaze and {Vendor} can be replaced by a company name or any other preferred name. {Platform} is a name and can be chosen as desired. After executing this command a folder will be created in the following path:

$PETALINUX/software/petalinux-dist/vendors/{Vendor}/{Platform}

The following configuration files are automatically generated and put in the above folder.

    config.arch
    config.device
    config.linux-2.6.x
    config.vendor
    {Vendor}-(Platform).dts
    config.mk
    xparameters.h

After navigating to the main directory of the XPS project, the following command must be executed:

    $ petalinux-copy-autoconfig

This command will rebuild the .DTS (Device Tree Source) based on the properties of the designed system.

The following two commands are for configuring the OS image kernel and applications:

    $ petalinux-config-kernel

    $ petalinux-config-apps
In the Kernel Configuration menu, under subsection Device Drivers, a change must be made for Graphic support in order for the system to support the Xilinx Frame Buffer controller. Figure A-10 shows how the menu should look like after the necessary changes have been made.

As can be seen in the image some options have been changed to “M” which represents a module to be added to the kernel.

In the Apps Configuration menu there is one modification to take place. Under the subsection System Settings, in the Kernel command line, the following line must be added:

```
mem=230M
```

This will force the system to only use the first 230 MB of the system memory. This will leave the rest of 26 MB untouched by kernel. In order to make the evaluation easier, the
A. System Design Procedure

author decided to specify a section in RAM for the designed IP Core and not deal with the DMA in kernel.

After this step, by executing the following command, the properties of the kernel image will be updated and it is ready to be built.

$ petalinux-platform-config –update

Now by executing the make command in Linux terminal, the cross compiling tool chain will create the OS image.

If everything goes fine the image can be downloaded to the board and boots up the system by the next command.

$ petalinux-jtag-boot –i images/image.elf

This is the last step of building the system. A developer can write shell scripts to automate the test execution, system setups and more just like what can be done on a Linux pc.

Details of how to compile required libraries for WebKit and the designed API are too long to be mentioned here. All the necessary files will be available at [35] for any interested reader or developer who desires to further improve this work.
Appendix B
System Design Codes

B.1 SYSTEM.UCF

# Virtex 5 ML505 Evaluation Platform
Net fpga_0_RS232_Uart_1_RX_pin LOC = AG15 | IOSTANDARD=LVCMS33;
Net fpga_0_RS232_Uart_1_TX_pin LOC = AG20 | IOSTANDARD=LVCMS33;
Net fpga_0_SRAM_Mem_A_pin<30> LOC=K12 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMS33;
Net fpga_0_SRAM_Mem_A_pin<29> LOC=K13 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMS33;
Net fpga_0_SRAM_Mem_A_pin<28> LOC=G23 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMS33;
Net fpga_0_SRAM_Mem_A_pin<27> LOC=J12 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMS33;
Net fpga_0_SRAM_Mem_A_pin<26> LOC=H12 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMS33;
Net fpga_0_SRAM_Mem_A_pin<25> LOC=J15 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMS33;
Net fpga_0_SRAM_Mem_A_pin<24> LOC=K16 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMS33;
Net fpga_0_SRAM_Mem_A_pin<23> LOC=K21 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMS33;
Net fpga_0_SRAM_Mem_A_pin<22> LOC=J22 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMS33;
Net fpga_0_SRAM_Mem_A_pin<21> LOC=L16 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMS33;
Net fpga_0_SRAM_Mem_A_pin<20> LOC=L20 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMS33;
Net fpga_0_SRAM_Mem_A_pin<19> LOC=L21 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMS33;
Net fpga_0_SRAM_Mem_A_pin<18> LOC=AE22 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMS33;
### B. System Design Codes

<table>
<thead>
<tr>
<th>Net</th>
<th>LOCO</th>
<th>SLEW</th>
<th>DRIVE</th>
<th>IOSTANDARD</th>
</tr>
</thead>
<tbody>
<tr>
<td>fpga_0_SRAM_Mem_A_pin&lt;8&gt;</td>
<td>AE12</td>
<td>FAST</td>
<td>8</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_A_pin&lt;7&gt;</td>
<td>AE13</td>
<td>FAST</td>
<td>8</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_OEN_pin LOC=J10</td>
<td>IOSTANDARD=LVDCI_33</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_WEN_pin LOC=AF20</td>
<td>SLEW = FAST</td>
<td>DRIVE = 12</td>
<td>LVCMOS33</td>
<td></td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_BEN_pin&lt;3&gt; LOC=J11</td>
<td>IOSTANDARD=LVDCI_33</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_BEN_pin&lt;2&gt; LOC=K11</td>
<td>IOSTANDARD=LVDCI_33</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_BEN_pin&lt;1&gt; LOC=D10</td>
<td>IOSTANDARD=LVDCI_33</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_BEN_pin&lt;0&gt; LOC=D11</td>
<td>IOSTANDARD=LVDCI_33</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_ADV_LDN_pin LOC=H8</td>
<td>IOSTANDARD=LVDCI_33</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;0&gt; LOC=AG22</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;1&gt; LOC=AH22</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;2&gt; LOC=AH12</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;3&gt; LOC=AG13</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;4&gt; LOC=AH20</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;5&gt; LOC=AH19</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;6&gt; LOC=AH14</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;7&gt; LOC=AH13</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;8&gt; LOC=AE16</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;9&gt; LOC=AE12</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;10&gt; LOC=AE13</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;11&gt; LOC=AD10</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;12&gt; LOC=AF20</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;13&gt; LOC=AE17</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;14&gt; LOC=AE19</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;15&gt; LOC=AE21</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;16&gt; LOC=J9</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;17&gt; LOC=K8</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;18&gt; LOC=K9</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;19&gt; LOC=B13</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;20&gt; LOC=C13</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;21&gt; LOC=G11</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;22&gt; LOC=G12</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_DQ_pin&lt;23&gt; LOC=M8</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;24&gt; LOC=L8</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;25&gt; LOC=F11</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;26&gt; LOC=E11</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;27&gt; LOC=M10</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;28&gt; LOC=L9</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;29&gt; LOC=E12</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;30&gt; LOC=E13</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_DQ_pin&lt;31&gt; LOC=N10</td>
<td>PULLDOWN</td>
<td>FAST</td>
<td>12</td>
<td>LVCMOS33</td>
</tr>
<tr>
<td>fpga_0_SRAM_Mem_EQPin LOC=G8</td>
<td>SLEW = FAST</td>
<td>DRIVE = 12</td>
<td>LVCMOS33</td>
<td></td>
</tr>
<tr>
<td>fpga_0_SRAM_EQPin LOC=AG21</td>
<td>IOSTANDARD=LVCMOS33;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
B. System Design Codes

Net fpga_0_Ethernet_MAC_PHY_tx_clk_pin LOC=K17 | IOSTANDARD = LVCMOS25;
Net fpga_0_Ethernet_MAC_PHY_rx_clk_pin LOC=H17 | IOSTANDARD = LVCMOS25;
Net fpga_0_Ethernet_MAC_PHY_crs_pin LOC=E34 | IOSTANDARD = LVCMOS25;
Net fpga_0_Ethernet_MAC_PHY_dv_pin LOC=E32 | IOSTANDARD = LVCMOS25;
Net fpga_0_Ethernet_MAC_PHY_rx_data_pin<0> LOC=A33 | IOSTANDARD = LVCMOS25;
Net fpga_0_Ethernet_MAC_PHY_rx_data_pin<1> LOC=B33 | IOSTANDARD = LVCMOS25;
Net fpga_0_Ethernet_MAC_PHY_rx_data_pin<2> LOC=C33 | IOSTANDARD = LVCMOS25;
Net fpga_0_Ethernet_MAC_PHY_rx_data_pin<3> LOC=C32 | IOSTANDARD = LVCMOS25;
Net fpga_0_Ethernet_MAC_PHY_col_pin LOC=B32 | IOSTANDARD = LVCMOS25;
Net fpga_0_Ethernet_MAC_PHY_rx_er_pin LOC=E33 | IOSTANDARD = LVCMOS25;
Net fpga_0_Ethernet_MAC_PHY_rst_n_pin LOC=J14 | IOSTANDARD = LVCMOS25 | TIG;
Net fpga_0_Ethernet_MAC_PHY_tx_en_pin LOC=AJ10 | IOSTANDARD = LVDCI_33;
Net fpga_0_Ethernet_MAC_PHY_tx_data_pin<3> LOC=AH10 | IOSTANDARD = LVDCI_33;
Net fpga_0_Ethernet_MAC_PHY_tx_data_pin<2> LOC=AH9 | IOSTANDARD = LVDCI_33;
Net fpga_0_Ethernet_MAC_PHY_tx_data_pin<1> LOC=AE11 | IOSTANDARD = LVDCI_33;
Net fpga_0_Ethernet_MAC_PHY_tx_data_pin<0> LOC=AF11 | IOSTANDARD = LVDCI_33;
Net fpga_0_Ethernet_MAC_PHY_MDC_pin LOC=H19 | IOSTANDARD = LVCMOS25;
Net fpga_0_Ethernet_MAC_PHY_MDIO_pin LOC=H13 | IOSTANDARD = LVCMOS25;
Net fpga_0_Ethernet_MAC_PHY_MDINT_pin LOC=H20 | IOSTANDARD = LVDCI_33;
B. System Design Codes

```
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<17> LOC=AB25 | IOSTANDARD = SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<18> LOC=AC27 | IOSTANDARD = SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<19> LOC=AA26 | IOSTANDARD = SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<20> LOC=AA24 | IOSTANDARD = SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<21> LOC=AB27 | IOSTANDARD = SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<22> LOC=AA25 | IOSTANDARD = SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<23> LOC=AC29 | IOSTANDARD = SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<24> LOC=AB26 | IOSTANDARD = SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<25> LOC=AB30 | IOSTANDARD = SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pins LOC=V30 | IOSTANDARD = SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pins LOC=V31 | IOSTANDARD = SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DM_pin<0> LOC=AJ31 | IOSTANDARD = SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DM_pin<1> LOC=AE28 | IOSTANDARD = SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DM_pin<2> LOC=AJ28 | IOSTANDARD = SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DM_pin<3> LOC=AB31 | IOSTANDARD = SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_pin<0> LOC=AA29 | IOSTANDARD = DIFF_SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_pin<1> LOC=AK28 | IOSTANDARD = DIFF_SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_pin<2> LOC=AK26 | IOSTANDARD = DIFF_SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_pin<3> LOC=AB31 | IOSTANDARD = DIFF_SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_pins LOC=V30 | IOSTANDARD = DIFF_SSTL18_I;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_pins LOC=V31 | IOSTANDARD = DIFF_SSTL18_I;
Net fpga_0_CLK_1_sys_clk_pin TMN_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 100000 kHz;
Net fpga_0_rst_1_sys_rst_pin TIG;
Net fpga_0_rst_1_sys_rst_pin LOC = E9 | IOSTANDARD=LVCMOS33 | PULLUP;

#### DDR2_SDRAM

# Define multicycle paths - these paths may take longer because additional
# time allowed for logic to settle in calibration/initialization FSM

# MUX Select for either rising/falling CLK0 for 2nd stage read capture
INST "/u_phy_calib_0/gen_rd_data_sel*.u_ff_rd_data_sel" TNM = "TNM_RD_DATA_SEL";
TIMESPEC "TS_MC_RD_DATA_SEL" = FROM "TNM_RD_DATA_SEL" TO FFS "TS_sys_clk_pin" * 2;
# MUX Select for read data - optional delay on data to account for byte skews
INST "/u_user_rd_0/*gen_rden_sel_mux*.u_ff_rden_sel_mux" TNM = "TNM_RDEN_SEL_MUX";
TIMESPEC "TS_MC_RDEN_SEL_MUX" = FROM "TNM_RDEN_SEL_MUX" TO FFS "TS_sys_clk_pin" * 2;
# Calibration/Initialization complete status flag (for PHY logic only)
INST "/u_phy_init_0/*ff_phy_init_data_sel" TNM = "TNM_PHY_INIT_DATA_SEL";
TIMESPEC "TS_MC_PHY_INIT_DATA_SEL_0" = FROM "TNM_PHY_INIT_DATA_SEL" TO FFS "TS_sys_clk_pin" * 2;
TIMESPEC "TS_MC_PHY_INIT_DATA_SEL_90" = FROM "TNM_PHY_INIT_DATA_SEL" TO FFS "TS_sys_clk_pin" * 2;
# Select [address] bits for SRL32 shift registers used in stage3/stage4
# calibration
INST "/u_phy_calib_0/*gen_gate_dly*.u_ff_gate_dly" TNM = "TNM_GATE_DLY";
TIMESPEC "TS_MC_GATE_DLY" = FROM "TNM_GATE_DLY" TO FFS "TS_sys_clk_pin" * 2;
INST "/u_phy_calib_0/*gen_rden_dly*.u_ff_rden_dly" TNM = "TNM_RDEN_DLY";
TIMESPEC "TS_MC_RDEN_DLY" = FROM "TNM_RDEN_DLY" TO FFS "TS_sys_clk_pin" * 2;
INST "/u_phy_calib_0/*gen_cal_rden_dly*.u_ff_cal_rden_dly" TNM = "TNM_CAL_RDEN_DLY";
TIMESPEC "TS_MC_CAL_RDEN_DLY" = FROM "TNM_CAL_RDEN_DLY" TO FFS "TS_sys_clk_pin" * 2;
```
### DQS Read Postamble Glitch Squelch circuit related constraints

#### LOC placement of DQS-squelch related IDDR and IDELAY elements

Each circuit can be located at any of the following locations:

1. Unused "N"-side of DQS diff pair I/O
2. DM data mask (output only, input side is free for use)
3. Any output-only site

```plaintext
INST "/gen_dqs[0].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y96";
INST "/gen_dqs[0].u_iob_dqs/u_iodelay_dq_ce" LOC = "IODELAY_X0Y96";
INST "/gen_dqs[1].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y58";
INST "/gen_dqs[1].u_iob_dqs/u_iodelay_dq_ce" LOC = "IODELAY_X0Y58";
INST "/gen_dqs[2].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y62";
INST "/gen_dqs[2].u_iob_dqs/u_iodelay_dq_ce" LOC = "IODELAY_X0Y62";
INST "/gen_dqs[3].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y100";
INST "/gen_dqs[3].u_iob_dqs/u_iodelay_dq_ce" LOC = "IODELAY_X0Y100";
```

```plaintext
#INST
"DDR2_SDRAM/DDR2_SDRAM/gen_no_iodelay_grp.gen_instantiate_idelayctrls[1].idelayctrl0" LOC = IDELAYCTRL_X0Y2;
#INST
"DDR2_SDRAM/DDR2_SDRAM/gen_no_iodelay_grp.gen_instantiate_idelayctrls[0].idelayctrl0" LOC = IDELAYCTRL_X0Y6;
#INST
"DDR2_SDRAM/DDR2_SDRAM/gen_no_iodelay_grp.gen_instantiate_idelayctrls[2].idelayctrl0" LOC = IDELAYCTRL_X0Y1;
```

#### LOC and timing constraints for flop driving DQS CE enable signal from fabric logic. Even though the absolute delay on this path is calibrated out (when synchronizing this output to DQS), the delay should still be kept as low as possible to reduce post-calibration voltage/temp variations - these are roughly proportional to the absolute delay of the path.

```plaintext
INST "/u_phy_calib_0/gen_gate[0].u_en_dqs_ff" LOC = SLICE_X0Y48;
INST "/u_phy_calib_0/gen_gate[1].u_en_dqs_ff" LOC = SLICE_X0Y29;
INST "/u_phy_calib_0/gen_gate[2].u_en_dqs_ff" LOC = SLICE_X0Y31;
INST "/u_phy_calib_0/gen_gate[3].u_en_dqs_ff" LOC = SLICE_X0Y50;
```

- Control for DQS gate - from fabric flop. Prevent "runaway" delay - two parts to this path: (1) from fabric flop to IDELAY, (2) from IDELAY to asynchronous reset of IDDR that drives the DQ CE's. A single number is used for all speed grades - value based on 333MHz. This can be relaxed for lower frequencies.

```plaintext
NET "/u_phy_io_0/en_dqs*" MAXDELAY = 600 ps;
NET "/u_phy_io_0/gen_dqs*.u_iob_dqs/en_dqs_sync" MAXDELAY = 850 ps;
```

#### "Half-cycle" path constraint from IDDR to CE pin for all DQ IDDR's for DQS Read Postamble Glitch Squelch circuit

```plaintext
INST "/gen_dqs[*].u_iob_dqs/u_iddr_dq_ce" TNM = "TNM_DQ_CE_IDDR";
```

### System Design Codes
B. System Design Codes

```
INST ")/gen_dq[*].u_iob_dq/gen_stg2_*.u_iddr_dq" TNM = "TNM_DQS_FLOPS";
TIMESPEC "TS_DQ_CE" = FROM "TNM_DQ_CE_IDDR" TO "TNM_DQS_FLOPS" 1.9 ns;

# IO Pad Location Constraints / Properties for TFT tft LCD Controller

NET xps_tft_0_TFT_IIC_SCL LOC = U27;
NET xps_tft_0_TFT_IIC_SCL LOC = T29;
NET xps_tft_0_TFT_IIC_SCL SLEW = SLOW;
NET xps_tft_0_TFT_IIC_SCL DRIVE = 6;
NET xps_tft_0_TFT_IIC_SCL TIG;
NET xps_tft_0_TFT_IIC_SCL IOSTANDARD = LVCMOS18 ; #ff LVCMOS33;
NET xps_tft_0_TFT_IIC_SCL LOC = U27;
NET xps_tft_0_TFT_IIC_SCL LOC = T29;
NET xps_tft_0_TFT_IIC_SCL SLEW = SLOW;
NET xps_tft_0_TFT_IIC_SCL DRIVE = 6;
NET xps_tft_0_TFT_IIC_SCL TIG;
NET xps_tft_0_TFT_IIC_SCL IOSTANDARD = LVCMOS18 ; #ff LVCMOS33;

B.2 SYSTEM.MHS

# #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#
# Created by Base System Builder Wizard for Xilinx EDK 12.4 Build EDK_MS4.81d
# Thu Nov 10 12:59:06 2011
# Target Board:  Xilinx Virtex 5 ML505 Evaluation Platform Rev 1
# Family:    virtex5
# Device:   xc5vlx50t
# Package:   ff1136
# Speed Grade:  -1
# Processor number: 1
# Processor 1: microblaze_0

NET xps_tft_0_TFT_DVI_DATA_pin<0> LOC = AB8;
NET xps_tft_0_TFT_DVI_DATA_pin<1> LOC = AC8;
NET xps_tft_0_TFT_DVI_DATA_pin<2> LOC = AN12;
NET xps_tft_0_TFT_DVI_DATA_pin<3> LOC = AP12;
NET xps_tft_0_TFT_DVI_DATA_pin<4> LOC = AA9;
NET xps_tft_0_TFT_DVI_DATA_pin<5> LOC = AA8;
NET xps_tft_0_TFT_DVI_DATA_pin<6> LOC = AM13;
NET xps_tft_0_TFT_DVI_DATA_pin<7> LOC = AN13;
NET xps_tft_0_TFT_DVI_DATA_pin<8> LOC = AA10;
NET xps_tft_0_TFT_DVI_DATA_pin<9> LOC = AB10;
NET xps_tft_0_TFT_DVI_DATA_pin<10> LOC = AP14;
NET xps_tft_0_TFT_DVI_DATA_pin<11> LOC = AN14;
NET xps_tft_0_TFT_DVI_DATA_pin<*> IOSTANDARD = LVDCI_33;

NET xps_tft_0_TFT_DVI_CLK_P_pin LOC = AL11;
NET xps_tft_0_TFT_DVI_CLK_P_pin IOSTANDARD = LVCMOS33 | DRIVE = 24 | SLEW = FAST;
NET xps_tft_0_TFT_DVI_CLK_N_pin LOC = AL10;
NET xps_tft_0_TFT_DVI_CLK_N_pin IOSTANDARD = LVCMOS33 | DRIVE = 24 | SLEW = FAST;

NET xps_tft_0_TFT_HSYNC_pin LOC = AM12;
NET xps_tft_0_TFT_HSYNC_pin IOSTANDARD = LVDCI_33;
NET xps_tft_0_TFT_VSYNC_pin LOC = AM11;
NET xps_tft_0_TFT_VSYNC_pin IOSTANDARD = LVDCI_33;
NET xps_tft_0_TFT_DE_pin LOC = AE8;
NET xps_tft_0_TFT_DE_pin IOSTANDARD = LVDCI_33;
NET xps_tft_0_reset_pin LOC = AK6;
NET xps_tft_0_reset_pin IOSTANDARD = LVCMOS33;
```
# System Design Codes

## 107

# System clock frequency: 125.0
# Debug Interface: On-Chip HW Debug Module

PARAMETER VERSION = 2.1.0

PORT fp
- ga_0_RS232_Uart_1_RX_pin = fpga_0_RS232_Uart_1_RX_pin, DIR = I
PORT fpga_0_RS232_Uart_1_TX_pin = fpga_0_RS232_Uart_1_TX_pin, DIR = O
PORT fpga_0_SRAM_Mem_A_pin = fpga_0_SRAM_Mem_A_pin_vslice_7_30_concat, DIR = O, VEC = [7:30]
PORT fpga_0_SRAM_Mem_CEN_pin = fpga_0_SRAM_Mem_CEN_pin, DIR = O
PORT fpga_0_SRAM_Mem_OEN_pin = fpga_0_SRAM_Mem_OEN_pin, DIR = O
PORT fpga_0_SRAM_Mem_WEN_pin = fpga_0_SRAM_Mem_WEN_pin, DIR = O
PORT fpga_0_SRAM_Mem_BEN_pin = fpga_0_SRAM_Mem_BEN_pin, DIR = O, VEC = [0:3]
PORT fpga_0_SRAM_Mem_ADV_LDN_pin = fpga_0_SRAM_Mem_ADV_LDN_pin, DIR = O
PORT fpga_0_SRAM_DQ_pin = fpga_0_SRAM_Mem_DQ_pin, DIR = O, VEC = [0:31]
PORT fpga_0_SRAM_CLK_OUT_pin = SRAM_CLK_OUT_s, DIR = I, SIGIS = CLK, CLK_FREQ = 125000000
PORT fpga_0_Ethernet_MAC_PHY_tx_clk_pin = fpga_0_Ethernet_MAC_PHY_tx_clk_pin, DIR = I
PORT fpga_0_Ethernet_MAC_PHY_rx_clk_pin = fpga_0_Ethernet_MAC_PHY_rx_clk_pin, DIR = I
PORT fpga_0_Ethernet_MAC_PHY_crs_pin = fpga_0_Ethernet_MAC_PHY_crs_pin, DIR = I
PORT fpga_0_Ethernet_MAC_PHY_dv_pin = fpga_0_Ethernet_MAC_PHY_dv_pin, DIR = I
PORT fpga_0_Ethernet_MAC_PHY_rx_data_pin = fpga_0_Ethernet_MAC_PHY_rx_data_pin, DIR = I, VEC = [3:0]
PORT fpga_0_Ethernet_MAC_PHY_col_pin = fpga_0_Ethernet_MAC_PHY_col_pin, DIR = I
PORT fpga_0_Ethernet_MAC_PHY_rst_n_pin = fpga_0_Ethernet_MAC_PHY_rst_n_pin, DIR = I
PORT fpga_0_Ethernet_MAC_PHY_rx_er_pin = fpga_0_Ethernet_MAC_PHY_rx_er_pin, DIR = I
PORT fpga_0_Ethernet_MAC_PHY_rst_n_pin = fpga_0_Ethernet_MAC_PHY_rst_n_pin, DIR = I
PORT fpga_0_Ethernet_MAC_PHY_rx_data_pin = fpga_0_Ethernet_MAC_PHY_rx_data_pin, DIR = I
PORT fpga_0_DDR2_SDRAM_DDR2_Clk_pin = fpga_0_DDR2_SDRAM_DDR2_Clk_pin, DIR = O, VEC = [1:0]
PORT fpga_0_DDR2_SDRAM_DDR2_Clk_n_pin = fpga_0_DDR2_SDRAM_DDR2_Clk_n_pin, DIR = O, VEC = [1:0]
PORT fpga_0_DDR2_SDRAM_DDR2_CE_pin = fpga_0_DDR2_SDRAM_DDR2_CE_pin, DIR = O, VEC = [1:0]
PORT fpga_0_DDR2_SDRAM_DDR2_CS_n_pin = fpga_0_DDR2_SDRAM_DDR2_CS_n_pin, DIR = O, VEC = [1:0]
PORT fpga_0_DDR2_SDRAM_DDR2_RAS_n_pin = fpga_0_DDR2_SDRAM_DDR2_RAS_n_pin, DIR = O
PORT fpga_0_DDR2_SDRAM_DDR2_CAS_n_pin = fpga_0_DDR2_SDRAM_DDR2_CAS_n_pin, DIR = O
PORT fpga_0_DDR2_SDRAM_DDR2_WE_n_pin = fpga_0_DDR2_SDRAM_DDR2_WE_n_pin, DIR = O
PORT fpga_0_DDR2_SDRAM_DDR2_BankAddr_pin = fpga_0_DDR2_SDRAM_DDR2_BankAddr_pin, DIR = O, VEC = [1:0]
PORT fpga_0_DDR2_SDRAM_DDR2_DQ_pin = fpga_0_DDR2_SDRAM_DDR2_DQ_pin, DIR = O, VEC = [3:0]
PORT fpga_0_DDR2_SDRAM_DDR2_DQS_n_pin = fpga_0_DDR2_SDRAM_DDR2_DQS_n_pin, DIR = O, VEC = [3:0]
PORT fpga_0_clk_1_sys_clk_pin = dcm_clk_s, DIR = I, SIGIS = CLK, CLK_FREQ = 1000000000
PORT fpga_0_rst_1_sys_rst_pin = sys_rst_s, DIR = I, SIGIS = RST, RST_POLARITY = 0
PORT xps_tft_0_TFT_HSYNC_pin = xps_tft_0_TFT_HSYNC, DIR = I
PORT xps_tft_0_TFT_VSYNC_pin = xps_tft_0_TFT_VSYNC, DIR = I
PORT xps_tft_0_TFT_DE_pin = xps_tft_0_TFT_DE, DIR = I
PORT xps_tft_0_TFT_DVI_CLK_P_pin = xps_tft_0_TFT_DVI_CLK_P, DIR = I
B. System Design Codes

PORT xps_tft_0_TFT_DVI_CLK_N_pin = xps_tft_0_TFT_DVI_CLK_N, DIR = O
PORT xps_tft_0_TFT_DVI_DATA_pin = xps_tft_0_TFT_DVI_DATA, DIR = O, VEC = [11:0]
PORT xps_tft_0_TFT_IIC_SCL = xps_tft_0_TFT_IIC_SCL, DIR = IO
PORT xps_tft_0_TFT_IIC_SDA = xps_tft_0_TFT_IIC_SDA, DIR = IO
PORT xps_tft_0_reset_pin = sys_periph_reset_n, DIR = O
PORT xps_tft_0_IP2INTC_Irpt_pin = xps_tft_0_IP2INTC_Irpt, DIR = O, SIGIS = INTERRUPT,
SENSITIVITY = EDGE_RISING
PORT xps_intc_0_Irq_pin = microblaze_0_Interrupt, DIR = O, SIGIS = INTERRUPT,
SENSITIVITY = EDGE_RISING

BEGIN microblaze
PARAMETER INSTANCE = microblaze_0
PARAMETER C_USE_BARREL = 1
PARAMETER C_DEBUG_ENABLED = 1
PARAMETER C_ICACHE_BASEADDR = 0x50000000
PARAMETER C_ICACHE_BYTE_SIZE = 16384
PARAMETER C_ICACHE_LINESIZE = 8
PARAMETER C_ICACHE_STREAMS = 1
PARAMETER C_ICACHE_VICTIMS = 8
PARAMETER C_DIV_HALF笥N_EXCEPTION = 1
PARAMETER C_DPLB_BUS_EXCEPTION = 1
PARAMETER C_IPLB_BUS_EXCEPTION = 1
PARAMETER C_ILL_OPCODE_EXCEPTION = 1
PARAMETER C_UNALIGNED_EXCEPTIONS = 1
PARAMETER C_USE_HW_MUL = 2
PARAMETER C_USE_DIV = 1
PARAMETER C_FSL_LINKS = 1
BUS_INTERFACE DEBUG = microblaze_0_mdm_bus
BUS_INTERFACE IICL = microblaze_0_IICL
BUS_INTERFACE DIXCL = microblaze_0_DXCL
BUS_INTERFACE IFLB = mb_plb
BUS_INTERFACE DPLB = mb_plb
BUS_INTERFACE DLMB = dlmb
BUS_INTERFACE ILMB = ilmb
PORT MB_RESET = mb_reset
PORT INTERRUPT = microblaze_0_Interrupt
END

BEGIN plb_v46
PARAMETER INSTANCE = mb_plb
PARAMETER HW_VER = 1.05.a
PORT PLB_CLK = clk_125_0000MHzPLL0
PORT SYS_Rst = sys_bus_reset
END

BEGIN lmb_v10
PARAMETER INSTANCE = ilmb
PARAMETER HW_VER = 1.00.a
PORT LMB_CLK = clk_125_0000MHzPLL0
PORT SYS_Rst = sys_bus_reset
END
B. System Design Codes

END

BEGIN lmb_v10
PARAMETER INSTANCE = dlmb
PARAMETER HW_VER = 1.00.a
PORT LMB_Clk = clk_125_0000MHzPLL0
PORT SYS_Rst = sys_bus_reset
END

BEGIN lmb_bram_if_cntlr
PARAMETER INSTANCE = dlmb_cntlr
PARAMETER HW_VER = 2.10.b
PARAMETER C_BASEADDR = 0x00000000
PARAMETER C_HIGHADDR = 0x00001fff
BUS_INTERFACE SLMB = dlmb
BUS_INTERFACE BRAM_PORT = dlmb_port
END

BEGIN lmb_bram_if_cntlr
PARAMETER INSTANCE = ilmb_cntlr
PARAMETER HW_VER = 2.10.b
PARAMETER C_BASEADDR = 0x00000000
PARAMETER C_HIGHADDR = 0x00001fff
BUS_INTERFACE SLMB = ilmb
BUS_INTERFACE BRAM_PORT = ilmb_port
END

BEGIN bram_block
PARAMETER INSTANCE = lmb_bram
PARAMETER HW_VER = 1.00.a
BUS_INTERFACE PORTA = ilmb_port
BUS_INTERFACE PORTB = dlmb_port
END

BEGIN xps_uartlite
PARAMETER INSTANCE = RS232_Uart_1
PARAMETER C_BAUDRATE = 115200
PARAMETER C_DATA_BITS = 8
PARAMETER C_USE_PARITY = 0
PARAMETER C_ODD_PARITY = 0
PARAMETER HW_VER = 1.01.a
PARAMETER C_BASEADDR = 0x84000000
PARAMETER C_HIGHADDR = 0x8400ffff
BUS_INTERFACE SPLB = mb_plb
PORT RX = fpga_0_RS232_Uart_1_RX_pin
PORT TX = fpga_0_RS232_Uart_1_TX_pin
PORT Interrupt = RS232_Uart_1_Interrupt
END

BEGIN xps_mch_emc
PARAMETER INSTANCE = SRAM
PARAMETER C_NUM_BANKS_MEM = 1
PARAMETER C_NUM_CHANNELS = 0
PARAMETER C_MEM0_WIDTH = 32
PARAMETER C_MAX_MEM_WIDTH = 32
PARAMETER C_INCLUDE_DATAWIDTH_MATCHING_0 = 0
PARAMETER C_SYNCH_MEM_0 = 1
PARAMETER C_TCEEDV_PS_MEM_0 = 0
PARAMETER C_TAVDV_PS_MEM_0 = 0
PARAMETER C_THZCE_PS_MEM_0 = 0
PARAMETER C_THZOE_PS_MEM_0 = 0
PARAMETER C_TWC_PS_MEM_0 = 0
PARAMETER C_TWP_PS_MEM_0 = 0
PARAMETER C_TLWE_PS_MEM_0 = 0

109
B. System Design Codes

PARAMETER HW_VER = 3.01.a
PARAMETER C_MEM0_BASEADDR = 0x86000000
PARAMETER C_MEM0_HIGHADDR = 0x860FFFFF
BUS_INTERFACE SPLB = mb_plb
PORT RdClk = clk_125_0000MHzPLL0
PORT Mem_A = 0b00000000 & fpga_0_SRAM_Mem_A_pin_vslice_7_30_concat & 0b0
PORT Mem_CEN = fpga_0_SRAM_Mem_CEN_pin
PORT Mem_OEN = fpga_0_SRAM_Mem_OEN_pin
PORT Mem_WEN = fpga_0_SRAM_Mem_WEN_pin
PORT Mem_BEN = fpga_0_SRAM_Mem_BEN_pin
PORT Mem_ADV_LDN = fpga_0_SRAM_Mem_ADV_LDN_pin
PORT Mem_DQ = fpga_0_SRAM_Mem_DQ_pin

BEGIN xps_ethernetlite
PARAMETER INSTANCE = Ethernet_MAC
PARAMETER HW_VER = 4.00.a
PARAMETER C_BASEADDR = 0x81000000
PARAMETER C_HIGHADDR = 0x8100ffff
BUS INTERFACE SPLB = mb_plb
PORT PHY_tx_clk = fpga_0_Ethernet_MAC_PHY_tx_clk_pin
PORT PHY_rx_clk = fpga_0_Ethernet_MAC_PHY_rx_clk_pin
PORT PHY_crs = fpga_0_Ethernet_MAC_PHY_crs_pin
PORT PHY_dv = fpga_0_Ethernet_MAC_PHY_dv_pin
PORT PHY_rx_data = fpga_0_Ethernet_MAC_PHY_rx_data_pin
PORT PHY_col = fpga_0_Ethernet_MAC_PHY_col_pin
PORT PHY_rx_er = fpga_0_Ethernet_MAC_PHY_rx_er_pin
PORT PHY_rst_n = fpga_0_Ethernet_MAC_PHY_rst_n_pin
PORT PHY_tx_en = fpga_0_Ethernet_MAC_PHY_tx_en_pin
PORT PHY_tx_data = fpga_0_Ethernet_MAC_PHY_tx_data_pin
PORT PHY_MDC = fpga_0_Ethernet_MAC_PHY_MDC_pin
PORT IP2INTC_Irpt = Ethernet_MAC_IP2INTC_Irpt
PORT PHY_MDIO = fpga_0_Ethernet_MAC_PHY_MDIO_pin

END

BEGIN mpmc
PARAMETER INSTANCE = DDR2_SDRAM
PARAMETER C_NUM_PORTS = 6
PARAMETER C_MEM_PARTNO = mt4htf3264h-53e
PARAMETER C_MEM_ODT_TYPE = 1
PARAMETER C_MEM_CLK_WIDTH = 2
PARAMETER C_MEM_ODT_WIDTH = 2
PARAMETER C_MEM_CE_WIDTH = 2
PARAMETER C_MEM_CS_N_WIDTH = 2
PARAMETER C_MEM_DATA_WIDTH = 32
PARAMETER C_DDR2_DQSN_ENABLE = 1
PARAMETER C_PIM0_BASETYPE = 2
PARAMETER C_PIM1_BASETYPE = 3
PARAMETER HW_VER = 6.02.a
PARAMETER C_SDMA1_P12L_CLK_RATIO = 1
PARAMETER C_PIM2_BASETYPE = 1
PARAMETER C_PIM3_BASETYPE = 1
PARAMETER C_PIM4_BASETYPE = 4
PARAMETER C_PIM4_DATA_WIDTH = 32
PARAMETER C_ALL_PIMS_SHARE_ADDRESSES = 1
PARAMETER C_MPMC_BASEADDR = 0x50000000
PARAMETER C_MPMC_HIGHADDR = 0x5FFFFFFF
PARAMETER C_SDMA_CTRL_BASEADDR = 0x84600000
PARAMETER C_SDMA_CTRL_HIGHADDR = 0x8460FFFF
PARAMETER C_PIM5_BASETYPE = 4
PARAMETER C_PIM5_DATA_WIDTH = 32
PARAMETER C_PIM6_BASETYPE = 0
BUS_INTERFACE SPLB0 = mb_plb
BUS_INTERFACE SDMA_CTRL1 = mb_plb
B. System Design Codes

BUS INTERFACE XCL2 = microblaze_0_IXCL
BUS INTERFACE XCL3 = microblaze_0_DXCL
BUS INTERFACE MPMC_PIM4 = displaymem2_0_XIL_NPI_Port1
PORT MPMC_Clk0 = clk_125_0000MHzPLL0
PORT MPMC_Clk0_DIV2 = clk_62_5000MHzPLL0
PORT MPMC_Clk90 = clk_125_0000MHz90PLL0
PORT MPMC_Clk_200MHz = clk_200_0000MHz
PORT MPMC_Rst = sys_periph_reset
PORT DDR2_Clk = fpga_0_DDR2_SDRAM_DDR2_Clk_pin
PORT DDR2_Clk_n = fpga_0_DDR2_SDRAM_DDR2_Clk_n_pin
PORT DDR2_CE = fpga_0_DDR2_SDRAM_DDR2_CE_pin
PORT DDR2_CS_n = fpga_0_DDR2_SDRAM_DDR2_CS_n_pin
PORT DDR2_ODT = fpga_0_DDR2_SDRAM_DDR2_ODT_pin
PORT DDR2_RAS_n = fpga_0_DDR2_SDRAM_DDR2_RAS_n_pin
PORT DDR2_CAS_n = fpga_0_DDR2_SDRAM_DDR2_CAS_n_pin
PORT DDR2_WE_n = fpga_0_DDR2_SDRAM_DDR2_WE_n_pin
PORT DDR2_BankAddr = fpga_0_DDR2_SDRAM_DDR2_BankAddr_pin
PORT DDR2.Addr = fpga_0_DDR2_SDRAM_DDR2.Addr_pin
PORT DDR2_DQ = fpga_0_DDR2_SDRAM_DDR2_DQ_pin
PORT DDR2_DQ_n = fpga_0_DDR2_SDRAM_DDR2_DQ_n_pin
PORT SDMA1_clk = clk_125_0000MHzPLL0

BEGIN xps_timer
PARAMETER INSTANCE = xps_timer_0
PARAMETER COUNT_WIDTH = 32
PARAMETER ONE_TIMER_ONLY = 0
PARAMETER HW_VER = 1.02.a
PARAMETER BASEADDR = 0x83c00000
PARAMETER HIGHADDR = 0x83c0ffff
BUS INTERFACE SPLB = mb_plb
PORT Interrupt = xps_timer_0_Interrupt
END

BEGIN clock_generator
PARAMETER INSTANCE = clock_generator_0
PARAMETER CLKIN_FREQ = 100000000
PARAMETER CLKOUT0_FREQ = 125000000
PARAMETER CLKOUT0_PHASE = 90
PARAMETER CLKOUT0_GROUP = PLL0
PARAMETER CLKOUT0_BUF = TRUE
PARAMETER CLKOUT1_FREQ = 125000000
PARAMETER CLKOUT1_PHASE = 0
PARAMETER CLKOUT1_GROUP = PLL0
PARAMETER CLKOUT1_BUF = TRUE
PARAMETER CLKOUT2_FREQ = 200000000
PARAMETER CLKOUT2_PHASE = 0
PARAMETER CLKOUT2_GROUP = NONE
PARAMETER CLKOUT2_BUF = TRUE
PARAMETER CLKOUT3_FREQ = 62500000
PARAMETER CLKOUT3_PHASE = 0
PARAMETER CLKOUT3_GROUP = PLL0
PARAMETER CLKOUT3_BUF = TRUE
PARAMETER CLKFBIN_FREQ = 125000000
PARAMETER CLKFBOUT_FREQ = 125000000
PARAMETER CLKFBOUT_BUF = TRUE
PARAMETER EXT_RESET_HIGH = 0
PARAMETER HW_VER = 4.01.a
PARAMETER CLKOUT4_FREQ = 25000000
PORT CLKIN = dcm_clk_s
PORT CLKOUT0 = clk_125_0000MHz90PLL0
PORT CLKOUT1 = clk_125_0000MHzPLL0
B. System Design Codes

PORT CLKOUT2 = clk_200_0000MHz
PORT CLKOUT3 = clk_62_5000MHzPLL0
PORT CLKFBIN = SRAM_CLK_FB_s
PORT CLKFBOUT = SRAM_CLK_OUT_s
PORT RST = sys_rst_s
PORT LOCKED = Dcm_all_locked
PORT CLKOUT4 = clk_25_0000MHz

BEGIN mdm
PARAMETER INSTANCE = mdm_0
PARAMETER C_MB_DBG_PORTS = 1
PARAMETER C_USE_UART = 1
PARAMETER HW_VER = 2.00.a
PARAMETER C_BASEADDR = 0x84400000
PARAMETER C_HIGHADDR = 0x84400000
BUS_INTERFACE SPLB = mb_plb
BUS_INTERFACE MBDEBUG_0 = microblaze_0_mdm_bus
PORT Debug_SYS_Rst = Debug_SYS_Rst
END

BEGIN proc_sys_reset
PARAMETER INSTANCE = proc_sys_reset_0
PARAMETER C_EXT_RESET_HIGH = 0
PARAMETER HW_VER = 2.00.a
PARAMETER C_BASEADDR = 0x84400000
PARAMETER C_HIGHADDR = 0x84400000
BUS_INTERFACE SPLB = mb_plb
PORT Slowest_sync_clk = clk_125_0000MHzPLL0
PORT Ext_Reset_In = sys_rst_s
PORT MB_Debug_Sys_Rst = Debug_SYS_Rst
PORT Dcm_locked = Dcm_all_locked
PORT MB_Reset = mb_reset
PORT Bus_Struct_Reset = sys_bus_reset
PORT Peripheral_Reset = sys_periph_reset
END

BEGIN xps_intc
PARAMETER INSTANCE = xps_intc_0
PARAMETER HW_VER = 2.01.a
PARAMETER C_BASEADDR = 0x81800000
PARAMETER C_HIGHADDR = 0x81800000
BUS_INTERFACE SPLB = mb_plb
PORT Intr = RS232_Uart_1_Interrupt & Ethernet_MAC_IP2INTC_Irpt & xps_timer_0_Interrupt & fpga_0_Ethernet_MAC_MDINT_pin & xps_tft_0_IP2INTC_Irpt
PORT Irq = microblaze_0_Interrupt
END

BEGIN xps_tft
PARAMETER INSTANCE = xps_tft_0
PARAMETER HW_VER = 2.01.a
PARAMETER C_DCR_SPLB_SLAVE_IF = 1
PARAMETER C_SPLB_BASEADDR = 0x86e00000
PARAMETER C_SPLB_HIGHADDR = 0x86e00000
PARAMETER C_TFT_INTERFACE = 1
PARAMETER C_I2C_SLAVE_ADDR = 0b1110110
PARAMETER C_DEFAULT_TFT_BASE_ADDR = 0x90000000
BUS_INTERFACE MPLB = mb_plb
BUS_INTERFACE SPLB = mb_plb
PORT TFT_HSYNC = xps_tft_0_TFT_HSYNC
PORT TFT_VSYNC = xps_tft_0_TFT_VSYNC
PORT TFT_CLAMP = xps_tft_0_TFT_CLAMP
PORT TFT_DVI_CLK_P = xps_tft_0_TFT_DVI_CLK_P
PORT TFT_DVI_CLK_N = xps_tft_0_TFT_DVI_CLK_N
PORT TFT_DVI_DATA = xps_tft_0_TFT_DVI_DATA
PORT TFT_IIC_SCL = xps_tft_0_TFT_IIC_SCL
PORT TFT_IIC_SDA = xps_tft_0_TFT_IIC_SDA

112
B. System Design Codes

PORT IP2INTC_Irpt = xps_tft_0_IP2INTC_Irpt
PORT SYS_TFT_Clk = clk_25_0000MHz
END

BEGIN util_vector_logic
PARAMETER INSTANCE = util_vector_logic_0
PARAMETER HW_VER = 1.00.a
PARAMETER C_OPERATION = not
PARAMETER C_SIZE = 1
PORT Op1 = sys_periph_reset
PORT Res = sys_periph_reset_n
END

BEGIN displaymem2
PARAMETER INSTANCE = displaymem2_0
PARAMETER HW_VER = 1.00.a
PARAMETER C_BASEADDR = 0xB6E00000
PARAMETER C_HIGHADDR = 0xB6E0FFFF
BUS_INTERFACE SPLB = mb_plb
BUS_INTERFACE XIL_NPI_Port1 = displaymem2_0_XIL_NPI_Port1
END

B.3 SYSTEM.MSS

PARAMETER VERSION = 2.2.0

BEGIN OS
PARAMETER OS_NAME = petalinux
PARAMETER OS_VER = 2.00.a
PARAMETER PROG_INSTANCE = microblaze_0
PARAMETER stdout = RS232_Uart_1
PARAMETER stdin = RS232_Uart_1
PARAMETER main_memory = DDR2_SDRAM
PARAMETER flash_memory = SRAM
PARAMETER lmb_memory = dlmb_cntlr
PARAMETER ethernet = Ethernet_MAC
PARAMETER timer = xps_timer_0
PARAMETER microblaze_exception_vectors =
((XEXC_NONE,XNullHandler,0),(XEXC_NONE,XNullHandler,0),(XEXC_NONE,XNullHandler,0),(XEXC_NONE,XNullHandler,0),
(XEXC_NONE,XNullHandler,0),(XEXC_NONE,XNullHandler,0),(XEXC_NONE,XNullHandler,0),(XEXC_NONE,XNullHandler,0))
END

BEGIN PROCESSOR
PARAMETER DRIVER_NAME = cpu
PARAMETER DRIVER_VER = 1.13.a
PARAMETER HW_INSTANCE = microblaze_0
PARAMETER COMPILER = mb-gcc
PARAMETER ARCHIVER = mb-ar
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = bram
PARAMETER DRIVER_VER = 2.00.a
PARAMETER HW_INSTANCE = dlmb_cntlr
END
BEGIN DRIVER
PARAMETER DRIVER_NAME = bram
PARAMETER DRIVER_VER = 2.00.a
PARAMETER HW_INSTANCE = ilmb_cntlr
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = generic
PARAMETER DRIVER_VER = 1.00.a
PARAMETER HW_INSTANCE = lmb_bram
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = uartlite
PARAMETER DRIVER_VER = 2.00.a
PARAMETER HW_INSTANCE = RS232_Uart_1
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = emc
PARAMETER DRIVER_VER = 3.01.a
PARAMETER HW_INSTANCE = SRAM
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = emaclite
PARAMETER DRIVER_VER = 3.01.a
PARAMETER HW_INSTANCE = Ethernet_MAC
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = mpmc
PARAMETER DRIVER_VER = 4.01.a
PARAMETER HW_INSTANCE = DDR2_SDRAM
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = tmrctr
PARAMETER DRIVER_VER = 2.02.a
PARAMETER HW_INSTANCE = xps_timer_0
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = generic
PARAMETER DRIVER_VER = 1.00.a
PARAMETER HW_INSTANCE = clock_generator_0
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = uartlite
PARAMETER DRIVER_VER = 2.00.a
PARAMETER HW_INSTANCE = mdm_0
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = generic
PARAMETER DRIVER_VER = 1.00.a
PARAMETER HW_INSTANCE = proc_sys_reset_0
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = intc
PARAMETER DRIVER_VER = 2.02.a
B. System Design Codes

PARAMETER HW_INSTANCE = xps_intc_0
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = tft
PARAMETER DRIVER_VER = 3.00.a
PARAMETER HW_INSTANCE = xps_tft_0
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = displaymem2
PARAMETER DRIVER_VER = 1.00.a
PARAMETER HW_INSTANCE = displaymem2_0
END
Appendix C

Text Display Engine IP Core

C.1 displaymem2.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library proc_common_v3_00_a;
use proc_common_v3_00_a.proc_common_pkg.all;
use proc_common_v3_00_a.ipif_pkg.all;

library plbv46_slave_single_v1_01_a;
use plbv46_slave_single_v1_01_a.plbv46_slave_single;
library displaymem2_v1_00_a;
use displaymem2_v1_00_a.user_logic;

-- Entity section

-- Definition of Generics:
-- C_BASEADDR -- PLBv46 slave: base address
-- C_HIGHADDR -- PLBv46 slave: high address
-- C_SPLB_ADDRWIDTH -- PLBv46 slave: address bus width
-- C_SPLB_NUM_MASTERS -- PLBv46 slave: Number of masters
-- C_SPLB_MID_WIDTH -- PLBv46 slave: master ID bus width
-- C_SPLB_NATIVE_DWIDTH -- PLBv46 slave: internal native data bus width
-- C_SPLB_P2P -- PLBv46 slave: point to point interconnect scheme
-- C_SPLB_SUPPORT_BURSTS -- PLBv46 slave: support bursts
-- C_SPLB_SIZEOF_SMALLEST_MASTER -- PLBv46 slave: width of the smallest master
-- C_SPLB_CLK_PERIOD_PS -- PLBv46 slave: bus clock in picoseconds
-- C_INCLUDE_DPHASE_TIMER -- PLBv46 slave: Data Phase Timer configuration; 0 = exclude timer, 1 = include timer
-- C_FAMILY -- Xilinx FPGA family

-- Definition of Ports:
-- SPLB_Clk -- PLB main bus clock
-- SPLB_Rst -- PLB main bus reset
-- PLB_Abus -- PLB address bus
-- PLB_UBus -- PLB upper address bus
-- PLB_PAValid -- PLB primary address valid indicator
-- PLB_SAValid -- PLB secondary address valid indicator
-- PLB_rdPrim -- PLB secondary to primary read request indicator
-- PLB_wrPrim -- PLB secondary to primary write request indicator
-- PLB_masterID -- PLB current master identifier
-- PLB_abort -- PLB abort request indicator
-- PLB_busLock -- PLB bus lock
-- PLB_RNW -- PLB read/not write
-- PLB_BE -- PLB byte enables
-- PLB_MSize -- PLB master data bus size
-- PLB_size -- PLB transfer size
-- PLB_type -- PLB transfer type
-- PLB_lockErr -- PLB lock error indicator
-- PLB_wrDBus -- PLB write data bus
-- PLB_wrBurst -- PLB burst write transfer indicator
-- PLB_rdBurst -- PLB burst read transfer indicator
-- PLB_wrPendReq -- PLB write pending bus request indicator
-- PLB_rdPendReq -- PLB read pending bus request indicator
-- PLB_wrPendPri -- PLB write pending request priority
-- PLB_rdPendPri -- PLB read pending request priority
-- PLB_reqPri -- PLB current request priority
-- PLB_TAttribute -- PLB transfer attribute
-- SL_addrAck -- Slave address acknowledge
-- SL_SSize -- Slave data bus size
-- SL_wait -- Slave wait indicator
-- SL_rearbitrate -- Slave re-arbitrate bus indicator
-- SL_wrAck -- Slave write data acknowledge
-- SL_wrComp -- Slave write transfer complete indicator
-- SL_wrTerm -- Slave terminate write burst transfer
-- SL_rdDBus -- Slave read data bus
-- SL_rdWdAddr -- Slave read word address
-- SL_rdAck -- Slave read data acknowledge
-- SL_rdComp -- Slave read transfer complete indicator
-- SL_rdTerm -- Slave terminate read burst transfer
-- SL_MBusy -- Slave busy indicator
-- SL_MWrErr -- Slave write error indicator
C. Text Display Engine IP Core

-- Sl_MRDErr -- Slave read error indicator
-- Sl_MIRQ  -- Slave interrupt indicator

entity displaymem2 is
  generic
  -- Bus protocol parameters, do not add to or delete
  C_BASEADDR : std_logic_vector := X"FFFFFFFF";
  C_HIGHADDR : std_logic_vector := X"00000000";
  C_SPLB_AWIDTH : integer := 32;
  C_SPLB_DWIDTH : integer := 128;
  C_SPLB_NUM_MASTERS : integer := 8;
  C_SPLB_NUM_MASTERS : integer := 8;
  C_SPLB_MID_WIDTH : integer := 3;
  C_SPLB_NATIVE_DWIDTH : integer := 32;
  C_SPLB_P2P : integer := 0;
  C_SPLB_SUPPORT_BURSTS : integer := 0;
  C_SPLB_SMALLEST_MASTER : integer := 32;
  C_SPLB_CLK_PERIOD_PS : integer := 10000;
  C_INCLUDE_DPHASE_TIMER : integer := 0;
  C_FAMILY : string := "virtex5";

  );
  port
  (
    XIL_NPI_Addr_Port1 : out std_logic_vector(0 to 31);
    XIL_NPI_AddrReq_Port1 : out std_logic;
    XIL_NPI_AddrAck_Port1 : in std_logic;
    XIL_NPI_RNW_Port1 : out std_logic;
    XIL_NPI_Size_Port1 : out std_logic_vector(0 to 3);
    XIL_NPI_RdFIFO_Data_Port1 : in std_logic_vector(0 to 31);
    XIL_NPI_WrFIFO_Data_Port1 : out std_logic_vector(0 to 31);
    XIL_NPI_WrFIFO_BE_Port1 : out std_logic;
    XIL_NPI_WrFIFO_Beck_Port1 : in std_logic;
    XIL_NPI_WriteFIFO_Push_Port1 : out std_logic;
    XIL_NPI_WriteFIFO_Pop_Port1 : in std_logic_vector(0 to 3);
    XIL_NPI_WriteFIFO_Eject_Push_Port1 : in std_logic;
    XIL_NPI_WriteFIFO_Flush_Port1 : in std_logic;
    XIL_NPI_WriteFIFO_Abort_Flush_Port1 : in std_logic;
    XIL_NPI_Watchdog_Port1 : in std_logic;
    XIL_NPI_InitDone_Port1 : in std_logic;
    -- Bus protocol ports, do not add to or delete
    SPLB_CLK : in std_logic;
    SPLB_RST : in std_logic;
    PLB_ABus : in std_logic_vector(0 to 31);
    PLB_ARBVal : in std_logic;
    PLB_ARBVal : in std_logic;
    PLB_rdPrim : in std_logic;
    PLB_wrPrim : in std_logic;
    PLB_masterID : in std_logic_vector(0 to C_SPLB_MID_WIDTH-1);
    PLB_abort : in std_logic;
    PLB_busLock : in std_logic;
    PLB_P2P : in std_logic;
    PLB_BE : in std_logic_vector(0 to C_SPLB_DWIDTH/8-1);
    PLB_MSIZ : in std_logic_vector(0 to 1);
    PLB_SIZE : in std_logic_vector(0 to 3);
    PLB_TYPE : in std_logic_vector(0 to 2);
    PLB_lockErr : in std_logic;
    PLB_wrDBus : in std_logic_vector(0 to C_SPLB_DWIDTH-1);
  );
C. Text Display Engine IP Core

PLB_wrBurst : in std_logic;
PLB_rdBurst : in std_logic;
PLB_wPendReq : in std_logic;
PLB_rPendReq : in std_logic;
PLB_wrPendPri : in std_logic_vector(0 to 1);
PLB_rdPendPri : in std_logic_vector(0 to 1);
PLB_TAttribute : in std_logic_vector(0 to 15);
Sl_addrAck : out std_logic;
Sl_SSize : out std_logic_vector(0 to 1);
Sl_wait : out std_logic;
Sl_rearbitrate : out std_logic;
Sl_wrDAck : out std_logic;
Sl_wrComp : out std_logic;
Sl_wrBTerm : out std_logic;
Sl_rdDBus : out std_logic_vector(C_SPLB_DWIDTH-1);
Sl_rdWdAddr : out std_logic_vector(0 to 3);
Sl_rdDAck : out std_logic;
Sl_rdComp : out std_logic;
Sl_rdBTerm : out std_logic;
Sl_MBusy : out std_logic_vector(C_SPLB_NUM_MASTERS-1);
Sl_MWrErr : out std_logic_vector(C_SPLB_NUM_MASTERS-1);
Sl_MRdErr : out std_logic_vector(C_SPLB_NUM_MASTERS-1);
Sl_MIRQ : out std_logic_vector(C_SPLB_NUM_MASTERS-1);

attribute SIGIS : string;
attribute SIGIS of SPLB_Clk : signal is "CLK";
attribute SIGIS of SPLB_Rst : signal is "RST";

end entity displaymem2;

-- Architecture section

architecture IMP of displaymem2 is

-- Array of base/high address pairs for each address range

constant ZERO_ADDR_PAD : std_logic_vector(0 to 31) := (others => '0');
constant USER_SLV_BASEADDR : std_logic_vector := C_BASEADDR;
constant USER_SLV_HIGHADDR : std_logic_vector := C_HIGHADDR;

constant IPIF_ARD_ADDR_RANGE_ARRAY : SLV64_ARRAY_TYPE :=
  (ZERO_ADDR_PAD & USER_SLV_BASEADDR, -- user logic slave space base address
   ZERO_ADDR_PAD & USER_SLV_HIGHADDR); -- user logic slave space high address

-- Array of desired number of chip enables for each address range

constant USER_SLV_NUM_REG : integer := 20;
constant USER_NUM_REG : integer := USER_SLV_NUM_REG;

constant IPIF_ARD_NUM_CE_ARRAY : INTEGER_ARRAY_TYPE :=
  (0 => pad_power2(USER_SLV_NUM_REG), -- number of ce for user logic slave space
   );

119
C. Text Display Engine IP Core

-- Ratio of bus clock to core clock (for use in dual clock systems)
-- 1 - ratio is 1:1
-- 2 - ratio is 2:1
-----------------------------------------------
constant IPIF_BUS2CORE_CLK_RATIO : integer := 1;
-----------------------------------------------

-- Width of the slave data bus (32 only)
------------------------------------------
constant USER_SLV_DWIDTH : integer := C_SPLB_NATIVE_DWIDTH;
constant IPIF_SLV_DWIDTH : integer := C_SPLB_NATIVE_DWIDTH;
------------------------------------------

-- Index for CS/CE
----------------------
constant USER_SLV_CS_INDEX : integer := 0;
constant USER_SLV_CE_INDEX := calc_start_ce_index(IPIF_ARD_NUM_CE_ARRAY, USER_SLV_CS_INDEX);
constant USER_CE_INDEX := USER_SLV_CE_INDEX;
----------------------

-- IP Interconnect (IPIC) signal declarations
---------------------------------------------
signal ipif_Bus2IP_Clk : std_logic;
signal ipif_Bus2IP_Reset : std_logic;
signal ipif_Bus2IP_Data : std_logic_vector(0 to IPIF_SLV_DWIDTH-1);
signal ipif_Bus2IP_WrAck : std_logic;
signal ipif_Bus2IP_RdAck : std_logic;
signal ipif_Bus2IP_Error : std_logic;
signal ipif_Bus2IP_Addr : std_logic_vector(0 to C_SPLB_AWIDTH-1);
signal ipif_Bus2IP_Data : std_logic_vector(0 to IPIF_SLV_DWIDTH-1);
signal ipif_Bus2IP_BE : std_logic_vector(0 to IPIF_SLV_DWIDTH/8-1);
signal ipif_Bus2IP_CS : std_logic_vector(0 to calc_num_ce(IPIF_ARD_NUM_CE_ARRAY)'length)/2-1);
signal ipif_Bus2IP_RdCE : std_logic_vector(0 to calc_num_ce(IPIF_ARD_NUM_CE_ARRAY)-1);
signal ipif_Bus2IP_WrCE : std_logic_vector(0 to calc_num_ce(IPIF_ARD_NUM_CE_ARRAY)-1);
signal user_Bus2IP_RdCE : std_logic_vector(0 to USER_NUM_REG-1);
signal user_Bus2IP_WrCE : std_logic_vector(0 to USER_NUM_REG-1);
signal user_Bus2IP_Data : std_logic_vector(0 to USER_SLV_DWIDTH-1);
signal user_Bus2IP_WrAck : std_logic;
signal user_Bus2IP_Error : std_logic;

begin

-- instantiate plbv46_slave_single

PLBV46_SLAVE_SINGLE_I : entity plbv46_slave_single_v1_01_a.plbv46_slave_single
generic map
(
C_ARD_ADDR_RANGE_ARRAY => IPIF_ARD_ADDR_RANGE_ARRAY,
C_ARD_NUM_CE_ARRAY => IPIF_ARD_NUM_CE_ARRAY,
C_SPLB_P2P => C_SPLB_P2P,
C_BUS2CORE_CLK_RATIO => IPIF_BUS2CORE_CLK_RATIO,
C_SPLB_MID_WIDTH => C_SPLB_MID_WIDTH,
C_SPLB_NUM_MASTERS => C_SPLB_NUM_MASTERS,
C_SPLB_AWIDTH => C_SPLB_AWIDTH,
C_SPLB_DWIDTH => C_SPLB_DWIDTH,
)
C. Text Display Engine IP Core

```vhdl
C_SIPIF_DWIDTH => IPIF_SLV_DWIDTH,
C_INCLUDE_DPHASE_TIMER => C_INCLUDE_DPHASE_TIMER,
C_FAMILY => C_FAMILY
)
port map
(
  SPLB_Clk => SPLB_Clk,
  SPLB_Rst => SPLB_Rst,
  PLB_ABus => PLB_ABus,
  PLB_UABus => PLB_UABus,
  PLB_FAValid => PLB_FAValid,
  PLB_SAValid => PLB_SAValid,
  PLB_rdPrim => PLB_rdPrim,
  PLB_wrPrim => PLB_wrPrim,
  PLB_masterID => PLB_masterID,
  PLB_abort => PLB_abort,
  PLB_busLock => PLB_busLock,
  PLB_RNW => PLB_RNW,
  PLB_BE => PLB_BE,
  PLB_MSize => PLB_MSize,
  PLB_size => PLB_size,
  PLB_type => PLB_type,
  PLB_lockErr => PLB_lockErr,
  PLB_wrDBus => PLB_wrDBus,
  PLB_wrBurst => PLB_wrBurst,
  PLB_rdBurst => PLB_rdBurst,
  PLB_wrPendReq => PLB.wrPendReq,
  PLB_rdPendReq => PLB.wrPendReq,
  PLB_wrPendPri => PLB.wrPendPri,
  PLB_rdPendPri => PLB.rdPendPri,
  PLB.reqPri => PLB.reqPri,
  PLB_TAttribute => PLB.TAttribute,
  Sl_addrAck => Sl_addrAck,
  Sl_SSsize => Sl_SSsize,
  Sl_wait => Sl_wait,
  Sl_rearbitrate => Sl_rearbitrate,
  Sl_wrAck => Sl_wrAck,
  Sl_wrComp => Sl_wrComp,
  Sl_wrBTerm => Sl_wrBTerm,
  Sl_rdDBus => Sl_rdDBus,
  Sl_rdWdAddr => Sl_rdWdAddr,
  Sl_rdDACK => Sl_rdDACK,
  Sl_rdComp => Sl_rdComp,
  Sl_rdBTerm => Sl_rdBTerm,
  Sl_Mbusy => Sl_Mbusy,
  Sl_MWrErr => Sl_MWrErr,
  Sl_MrdErr => Sl_MrdErr,
  Sl_MIRQ => Sl_MIRQ,
  BusZIP_Clk => ipif.BusZIP_Clk,
  BusZIP_Reset => ipif.BusZIP_Reset,
  IP2Bus_Data => ipif.IP2Bus_Data,
  IP2Bus_WrAck => ipif.IP2Bus_WrAck,
  IP2Bus_RdAck => ipif.IP2Bus_RdAck,
  IP2Bus_Error => ipif.IP2Bus_Error,
  BusZIP_Addr => ipif.BusZIP_Addr,
  BusZIP_Data => ipif.BusZIP_Data,
  BusZIP_RNW => ipif.BusZIP_RNW,
  BusZIP_BE => ipif.BusZIP_BE,
  BusZIP_CS => ipif.BusZIP_CS,
  BusZIP_RdCE => ipif.BusZIP_RdCE,
  BusZIP_WrCE => ipif.BusZIP_WrCE
);
```

-- instantiate User Logic
C. Text Display Engine IP Core

USER_LOGIC_I : entity displaymem2_v1_00_a.user_logic
    generic map
    (    
        C_SLV_DWIDTH => USER_SLV_DWIDTH,
        C_NUM_REG => USER_NUM_REG
    )
    port map
    (    
        XIL_NPI_Addr_Port1 => XIL_NPI_Addr_Port1,
        XIL_NPI_AddrReq_Port1 => XIL_NPI_AddrReq_Port1,
        XIL_NPI_AddrAck_Port1 => XIL_NPI_AddrAck_Port1,
        XIL_NPI_RNW_Port1 => XIL_NPI_RNW_Port1,
        XIL_NPI_Size_Port1 => XIL_NPI_Size_Port1,
        XIL_NPI_RdModWr_Port1 => XIL_NPI_RdModWr_Port1,
        XIL_NPI_WrFIFO_Data_Port1 => XIL_NPI_WrFIFO_Data_Port1,
        XIL_NPI_WrFIFO_BE_Port1 => XIL_NPI_WrFIFO_BE_Port1,
        XIL_NPI_WrFIFO_Push_Port1 => XIL_NPI_WrFIFO_Push_Port1,
        XIL_NPI_RdFIFO_Data_Port1 => XIL_NPI_RdFIFO_Data_Port1,
        XIL_NPI_RdFIFO_Pop_Port1 => XIL_NPI_RdFIFO_Pop_Port1,
        XIL_NPI_RdFIFO_RdWdAddr_Port1 => XIL_NPI_RdFIFO_RdWdAddr_Port1,
        XIL_NPI_WrFIFO_Empty_Port1 => XIL_NPI_WrFIFO_Empty_Port1,
        XIL_NPI_WrFIFO_AlmostFull_Port1 => XIL_NPI_WrFIFO_AlmostFull_Port1,
        XIL_NPI_WrFIFO_Flush_Port1 => XIL_NPI_WrFIFO_Flush_Port1,
        XIL_NPI_RdFIFO_Empty_Port1 => XIL_NPI_RdFIFO_Empty_Port1,
        XIL_NPI_RdFIFO_AlmostFull_Port1 => XIL_NPI_RdFIFO_AlmostFull_Port1,
        XIL_NPI_RdFIFO_Flush_Port1 => XIL_NPI_RdFIFO_Flush_Port1,
        XIL_NPI_InitDone_Port1 => XIL_NPI_InitDone_Port1,
    );

    -- MAP USER PORTS ABOVE THIS LINE -----------------------

    Bus2IP_Clk => ipif_Bus2IP_Clk,
    Bus2IP_Reset => ipif_Bus2IP_Reset,
    Bus2IP_Data => ipif_Bus2IP_DATA,
    Bus2IP_BE => ipif_Bus2IP_BE,
    Bus2IP_RdCE => user_Bus2IP_RdCE,
    Bus2IP_WrCE => user_Bus2IP_WrCE,
    IP2Bus_Data => user_IP2Bus_Data,
    IP2Bus_RdAck => user_IP2Bus_RdAck,
    IP2Bus_WrAck => user_IP2Bus_WrAck,
    IP2Bus_Error => user_IP2Bus_Error;

    -- connect internal signals

    if if IP2Bus_Data <= user IP2Bus_Data;
    if if IP2Bus_WrAck <= user IP2Bus_WrAck;
    if if IP2Bus_RdAck <= user IP2Bus_RdAck;
    if if IP2Bus_Error <= user IP2Bus_Error;

    end_IMP;
C.2 user_logic.vhd

-- Hardware Text Layout Design and Display Engine

-- ***************************************************************************
-- ** Copyright (c) 2013 Soheil Servati Beiragh, All rights reserved. **
-- ** UNIVERSITY of WINDSOR **
-- ** ***************************************************************************
-- Filename: user_logic.vhd
-- Version: 1.00.a
-- VHDL Standard: VHDL'93

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
library proc_common_v3_00_a;
use proc_common_v3_00_a.proc_common_pkg.all;

entity user_logic is
  generic
  (
    C_SLV_DWIDTH : integer := 32;
    C_NUM_REG   : integer := 20
  );
  port
  (
    XIL_NPI_Addr_Port1: out std_logic_vector(0 to 31);
    XIL_NPI_AddrReq_Port1: out std_logic;
  );
end entity user_logic;
C. Text Display Engine IP Core

XIL_NPI_AddrAck_Port1: in std_logic;
XIL_NPI_RNW_Port1: out std_logic;
XIL_NPI_Size_Port1: out std_logic_vector(0 to 3);
XIL_NPI_RdMoWr_Port1: out std_logic;
XIL_NPI_WrFIFO_Data_Port1: out std_logic_vector(0 to 3);
XIL_NPI_WrFIFO_Push_Port1: out std_logic;
XIL_NPI_RdFIFO_Data_Port1: in std_logic_vector(0 to 31);
XIL_NPI_RdFIFO_Pop_Port1: out std_logic;
XIL_NPI_WrFIFO_Empty_Port1: in std_logic;
XIL_NPI_WrFIFO_AlmostFull_Port1: in std_logic;
XIL_NPI_WrFIFO_Flush_Port1: out std_logic;
XIL_NPI_RdFIFO_Empty_Port1: in std_logic;
XIL_NPI_RdFIFO_Flush_Port1: out std_logic;
XIL_NPI_RdFIFO_Latency_Port1: in std_logic_vector(0 to 1);
XIL_NPI_InitDone_Port1: in std_logic;

-- Bus protocol ports, do not add to or delete
Bus2IP_Clk : in std_logic;
Bus2IP_Reset : in std_logic;
Bus2IP_Data : in std_logic_vector(0 to C_SLV_DWIDTH-1);
Bus2IP_BE : in std_logic_vector(0 to C_NUM_REG-1);
Bus2IP_RdCE : in std_logic;
Bus2IP_WrCE : in std_logic;
IP2Bus_Data : out std_logic_vector(0 to C_SLV_DWIDTH-1);
IP2Bus_RdAck : out std_logic;
IP2Bus_WrAck : out std_logic;
IP2Bus_Error : out std_logic;

attribute SIGIS : string;
attribute SIGIS of Bus2IP_Clk : signal is "CLK";
attribute SIGIS of Bus2IP_Reset : signal is "RST";
attribute use_dsp48 : string;
attribute use_dsp48 of user_logic : entity is "no";
end entity user_logic;

-- Architecture section

architecture IMP of user_logic is

-- Constant Memory Addresses are based on Xilinx XUPV5-LX110T and Sample API,
-- To use with other kits these addresses must be adjusted!
constant KADDRESS : integer := 1584954888;
constant SKADDRESS : integer := 1584954944;
constant SLADDRESS : integer := 1584529408;
constant s_indexarray : integer := 1584398336;
constant s_SAddress : integer := 1584922624;
constant s_charprop : integer := 1584463827;

type vector_arr is array(0 to 99) of std_logic_vector(0 to 31);
type integer_arr is array(0 to 99) of integer;
C. Text Display Engine IP Core

type state_type is (IDLE, CHARPO, CHARP1, LAYTS0, LAYTS1, LAYSC, LAYTS2, LAYTS3, LAYTS4, LAYTS5, LAYTS6, LAYTS7, LAYST, LAYTS8, LAYTS9, LAYTS10, LAYTS11, KERNL0, KERNL1, KERNL2, KERNL3, KERNL4, KERNL5, KERNL6, KERNL7, KERNL8, KERNL9, KERNL10, KERNL11, KERNL12, KERNL13, KERNL14, KERNL15, OUTPUT0, OUTPUT1, OUTPUT2, OUTPUT3, OUTPUT4, OUTPUT5, INITLC, INITL0, INITL1, INITL2, INITL3, INITL4, CLEANC, CLEAN0, CLEAN1, CLEAN2, CLEAN3, CLEAN4, REPEAT, READA0, READA1, READP0, LBADD0, LBADD1, READP1, READP2, READP3, READP4, CHECKS, READPE, START0, START1, READ0, READ1, READ2, READ3, READ4, WRITE0, WRITE1, WRITE2, WRITE3, WRITE4, WRITE5, WRITE6, WRITE7, WRITE8, WRITE9, WRITE10, WRITE11, DONE);

signal status_Port1, status_Return : state_type;
signal ran_once, LastLine, INITCHECK : boolean;
signal s_E, s_Rst : std_logic;
signal s_FAddress, s_RAddress, s_AvailableWidth, s_xpos, s_ypos, s_TextLength, s_MaxWidth, s_MaxHeight, s_Align, s_Done : std_logic_vector(0 to 31);
signal s_endArray, s_maxhArray, s_BAddress, s_SRow, s_BHeight, s_SLine : std_logic_vector(0 to 31);
signal tempData, midData, s_cycles : std_logic_vector(0 to 31);
signal CYCLES : std_logic_vector(31 downto 0);

-- Added for caching character properties
signal charindex, charheight, charpitch, charadvance, charhoriBY, chardiff, spacepitch : integer;
signal CharPropArray : vector_arr;

-- Counters
signal C1, R1, R2, R3, R2MB, L1, LN, CLoop1, CLoop2, kern, K1 : integer;

-- Added for Layout Calculations
signal prevcharindex, nextcharindex : integer;
signal p_space, SpaceLength, SLCounter, scheck, LLCounter, LCounter, TLLCounter : integer;
signal WordLength, SMaxLineHeight, MaxHeight, MaxBearing, MaxUnder : integer;

-- Added for Kerning Calculations
signal startRow, endRow_gindex, endRow_nextgindex, endRow, gRow, ngRow, kernSum, RK, Min, HighBearingY : integer;
signal nextcharheight, nextcharpitch, nextcharhoriBY : integer;

-- Added for Text Placement
signal maxwidth, maxBheight, ypos, xpos, SRow, BHeight, SLine, templ, temph, SRBL, cur_ypos : integer;
signal Alignment, TextLength, AvailableWidth : integer;
signal FAddress, RAddress, SAddress, endline, maxline, sindexline : integer;
signal MaxLineHeight, EndLIndex, LengthIndex, StartIndex, SpaceCount, LengthIndexS : integer;

-- Signals for user logic s/w accessible register

signal slv_reg0 : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg1 : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg2 : std_logic_vector(0 to C_SLV_DWIDTH-1);
C. Text Display Engine IP Core

```vhdl
signal slv_reg3 : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg4 : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg5 : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg6 : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg7 : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg8 : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg9 : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg10 : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg11 : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg12 : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg13 : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg14 : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg15 : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg16 : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg17 : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal slv_reg18 : std_logic_vector(0 to C_SLV_DWIDTH-1);

begin

  -- USER logic implementation

  slv_reg_write_sel <= Bus2IP_WrCE(0 to 19);
  slv_reg_read_sel <= Bus2IP_rdCE(0 to 19);
  slv_write_ack <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1) or Bus2IP_WrCE(2) or
                   Bus2IP_WrCE(3) or Bus2IP_WrCE(4) or Bus2IP_WrCE(5) or Bus2IP_WrCE(6) or
                   Bus2IP_WrCE(7) or Bus2IP_WrCE(8) or Bus2IP_WrCE(9) or Bus2IP_WrCE(10) or
                   Bus2IP_WrCE(11) or Bus2IP_WrCE(12) or
                   Bus2IP_WrCE(13) or Bus2IP_WrCE(14) or Bus2IP_WrCE(15) or Bus2IP_WrCE(16) or
                   Bus2IP_WrCE(17) or Bus2IP_WrCE(18) or Bus2IP_WrCE(19);
  slv_read_ack <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1) or Bus2IP_RdCE(2) or
                 Bus2IP_RdCE(3) or Bus2IP_RdCE(4) or Bus2IP_RdCE(5) or Bus2IP_RdCE(6) or
                 Bus2IP_RdCE(7) or Bus2IP_RdCE(8) or Bus2IP_RdCE(9) or Bus2IP_RdCE(10) or
                 Bus2IP_RdCE(11) or Bus2IP_RdCE(12) or
                 Bus2IP_RdCE(13) or Bus2IP_RdCE(14) or Bus2IP_RdCE(15) or Bus2IP_RdCE(16) or
                 Bus2IP_RdCE(17) or Bus2IP_RdCE(18) or Bus2IP_RdCE(19);

  -- implement slave model software accessible register(s)
  SLAVE_REG_WRITE_PROC : process (Bus2IP_Clk) is
  begin
    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
      if Bus2IP_Reset = '1' then
        slv_reg0 <= (others => '0');
        slv_reg1 <= (others => '0');
        slv_reg2 <= (others => '0');
        slv_reg3 <= (others => '0');
        slv_reg4 <= (others => '0');
        slv_reg5 <= (others => '0');
        slv_reg6 <= (others => '0');
        slv_reg7 <= (others => '0');
        slv_reg8 <= (others => '0');
        slv_reg9 <= (others => '0');
        slv_reg10 <= (others => '0');
        slv_reg11 <= (others => '0');
        slv_reg12 <= (others => '0');
        slv_reg13 <= (others => '0');
        slv_reg14 <= (others => '0');
        slv_reg15 <= (others => '0');
        slv_reg16 <= (others => '0');
        slv_reg17 <= (others => '0');
        slv_reg18 <= (others => '0');
      else

        signal slv_reg_write_sel : std_logic_vector(0 to C_SLV_DWIDTH-1);
        signal slv_reg_read_sel : std_logic_vector(0 to C_SLV_DWIDTH-1);
        signal slv_reg_write_ack : std_logic;
C. Text Display Engine IP Core

```vhdl
-- implement slave model software accessible register(s) read mux
SLAVE_REG_WRITE_PROC : process(slv_reg_write_sel, slv_reg0, slv_reg1, slv_reg2, slv_reg3, slv_reg4, slv_reg5, slv_reg6, slv_reg7, slv_reg8, slv_reg9, slv_reg10, slv_reg11, slv_reg12, slv_reg13, slv_reg14, slv_reg15, slv_reg16, slv_reg17, slv_reg18, slv_reg19) is
begin
  case slv_reg_write_sel is
    when "100000000000000000000000" => slv_reg0 <= Bus2IP_Data(0 to C_SLV_DWIDTH-1);
    when "010000000000000000000000" => slv_reg1 <= Bus2IP_Data(0 to C_SLV_DWIDTH-1);
    when "001000000000000000000000" => slv_reg2 <= Bus2IP_Data(0 to C_SLV_DWIDTH-1);
    when "000100000000000000000000" => slv_reg3 <= Bus2IP_Data(0 to C_SLV_DWIDTH-1);
    when "000010000000000000000000" => slv_reg4 <= Bus2IP_Data(0 to C_SLV_DWIDTH-1);
    when "000001000000000000000000" => slv_reg5 <= Bus2IP_Data(0 to C_SLV_DWIDTH-1);
    when "000000100000000000000000" => slv_reg6 <= Bus2IP_Data(0 to C_SLV_DWIDTH-1);
    when "000000010000000000000000" => slv_reg7 <= Bus2IP_Data(0 to C_SLV_DWIDTH-1);
    when "000000001000000000000000" => slv_reg8 <= Bus2IP_Data(0 to C_SLV_DWIDTH-1);
    when "000000000100000000000000" => slv_reg9 <= Bus2IP_Data(0 to C_SLV_DWIDTH-1);
    when "000000000010000000000000" => slv_reg10 <= Bus2IP_Data(0 to C_SLV_DWIDTH-1);
    when "000000000001000000000000" => slv_reg11 <= Bus2IP_Data(0 to C_SLV_DWIDTH-1);
    when "000000000000100000000000" => slv_reg12 <= Bus2IP_Data(0 to C_SLV_DWIDTH-1);
    when "000000000000010000000000" => slv_reg13 <= Bus2IP_Data(0 to C_SLV_DWIDTH-1);
    when "000000000000001000000000" => slv_reg14 <= Bus2IP_Data(0 to C_SLV_DWIDTH-1);
    when "000000000000000100000000" => slv_reg15 <= Bus2IP_Data(0 to C_SLV_DWIDTH-1);
    when "000000000000000010000000" => slv_reg16 <= Bus2IP_Data(0 to C_SLV_DWIDTH-1);
    when "000000000000000001000000" => slv_reg17 <= Bus2IP_Data(0 to C_SLV_DWIDTH-1);
    when "000000000000000000100000" => slv_reg18 <= Bus2IP_Data(0 to C_SLV_DWIDTH-1);
    when others => null;
  end case;
end if;
end if;
end process SLAVE_REG_WRITE_PROC;
```

displaymem: process (Bus2IP_Clk) is
begin
  if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
    CYCLES <= CYCLES + "0000000000000000000000000000000001"
  end if;
  if s_Rst = '0' then
```
```
C. Text Display Engine IP Core

XIL_NPI_Addr_Port1 <= "00000000000000000000000000000000";
XIL_NPI_AddReq_Port1 <= '0';
XIL_NPI_RNW_Port1 <= '0';
XIL_NPI_Size_Port1 <= "0000";
XIL_NPI_RdModWr_Port1 <= '0';
XIL_NPI_WrFIFO_Push_Port1 <= '0';
XIL_NPI_WrFIFO_Flush_Port1 <= '0';
XIL_NPI_WrFIFO_Data_Port1 <= "00000000000000000000000000000000";
s_Done <= "00000000000000000000000000000000";
ran_once <= false;
status_Port1 <= IDLE;
else
  case (status_Port1) is
    when IDLE =>
      if (s_E = '1' and ran_once = false) then
        C1 <= 0;
        R1 <= 0;
        R2 <= 0;
        R3 <= 0;
        L1 <= 0;
        LN <= 0;
        K1 <= 0;
        LCounter <= 0;
        LLCounter <= 0;
        SLCounter <= 0;
        TLLCounter <= 0;
        MaxBearing <= 0;
        MaxUnder <= 0;
        MaxHeight <= 0;
        WordLength <= 0;
        SpaceLength <= 0;
        temp <= 0;
        temp <= 0;
        xpos <= TO_INTEGER(unsigned (s_xpos));
        ypos <= TO_INTEGER(unsigned (s_ypos));
        cur_ypos <= 0;
        INITCHECK <= false;
        s_Done <= "00000000000000000000000000000000";
        status_Port1 <= CHARP0;
      elsif (s_E = '0') then
        ran_once <= false;
        status_Port1 <= IDLE;
      end if;
    -- Loading properties of characters (Caching)!
    when CHARP0 =>
      if (C1 < 100) then
        status_Return <= CHARP1;
        RAddress <= s_charprop + (TO_INTEGER(to_unsigned (C1,32) sll 2));
        status_Port1 <= READ0;
      else
        spacepitch <= TO_INTEGER(unsigned (CharPropArray(C1)(24 to 31)));
        status_Port1 <= LAYTS0;
      end if;
    when CHARP1 =>
      CharPropArray(C1)(0 to 7) <= midData(0 to 7);
      CharPropArray(C1)(8 to 15) <= midData(8 to 15);
      CharPropArray(C1)(16 to 23) <= midData(16 to 23);
      CharPropArray(C1)(24 to 31) <= midData(24 to 31);
      C1 <= C1 + 1;
      LengthIndex <= 0;
end case;
C. Text Display Engine IP Core

StartIndex <= 0;
status_Port1 <= CHARP0;

-- Calculating Layout (End of Lines and Line Heights)
when LAYTS0 =>
  LengthIndex <= 0;
  if (L1 < TextLength) then
    status_Return <= LAYTS1;
    RAddress <= s_indexarray + (TO_INTEGER(to_unsigned (L1,32) sll 2));
  LastLine <= false;
  status_Port1 <= READ0;
else
  LengthIndex <= LengthIndex + AvailableWidth + LLCounter + WordLength + SpaceLength;
  MaxLineHeight <= MaxBearing + MaxUnder;
  EndLIndex <= TextLength;
  LastLine <= true;
  status_Port1 <= LAYTS9;
end if;
when LAYTS1 =>
  charindex <= TO_INTEGER(unsigned (tempData));
  status_Port1 <= LAYTS;
when LAYTS3 =>
  charpitch <= TO_INTEGER(unsigned (CharPropArray(charindex)(24 to 31)));
  charheight <= TO_INTEGER(unsigned (CharPropArray(charindex)(16 to 23)));
  charhoriBY <= TO_INTEGER(unsigned (CharPropArray(charindex)(8 to 15)));
  charadvance <= TO_INTEGER(unsigned (CharPropArray(charindex)(8 to 15)) - TO_INTEGER(unsigned (CharPropArray(charindex)(16 to 23)));
  status_Port1 <= LAYTS2;
when LAYTS2 =>
  if (charindex = 3) then
    status_Return <= LAYTS3;
    RAddress <= s_indexarray + (TO_INTEGER(to_unsigned ((L1-1),32) sll 2));
  else
    status_Port1 <= READ0;
  end if;
when LAYTS3 =>
  prevcharindex <= TO_INTEGER(unsigned (tempData));
  status_Port1 <= LAYTS4;
when LAYTS4 =>
  if (prevcharindex = 3) then
    SpaceLength <= SpaceLength + spacepitch;
  else
    p_space <= L1;
    SpaceLength <= spacepitch;
  end if;
  SpaceCount <= SpaceCount + 1;
  WordLength <= 0;
  status_Port1 <= LAYTS5;
when LAYTS5 =>
  status_Return <= LAYTS6;
  RAddress <= s_indexarray + (TO_INTEGER(to_unsigned ((L1+1),32) sll 2));
  status_Port1 <= READ0;
when LAYTS6 =>
nextcharindex <= TO_INTEGER(unsigned {tempData});
if (charindex = 3) then
  status_Port1 <= KERNL12;
else
  status_Port1 <= KERNL0;
end if;

-- Calculating Boundries for Kerning Calculation
when KERNL0 =>
  if (nextcharindex = 3) then
    status_Port1 <= KERNL12;
  else
    kern <= 0;
    nextcharpitch <= TO_INTEGER(unsigned (CharPropArray(nextcharindex)(24 to 31)));
    nextcharheight <= TO_INTEGER(unsigned (CharPropArray(nextcharindex)(16 to 23)));
    nextcharhoriBY <= TO_INTEGER(unsigned (CharPropArray(nextcharindex)(8 to 15)));
    Min <= charpitch + TO_INTEGER(unsigned (CharPropArray(nextcharindex)(24 to 31)));
    status_Port1 <= KERNL1;
  end if;
when KERNL1 =>
  if (charhoriBY > nextcharhoriBY) then
    HighBearingY <= charhoriBY;
    startRow <= charhoriBY - nextcharhoriBY;
  else
    HighBearingY <= nextcharhoriBY;
    startRow <= nextcharhoriBY - charhoriBY;
  end if;
  status_Port1 <= KERNL2;
when KERNL2 =>
  endRow_gindex <= HighBearingY - charhoriBY + charheight;
  endRow_nextgindex <= HighBearingY - nextcharhoriBY + nextcharheight;
  status_Port1 <= KERNL3;
when KERNL3 =>
  if (endRow_nextgindex < endRow_gindex) then
    endRow <= endRow_nextgindex;
  else
    endRow <= endRow_gindex;
  end if;
  status_Port1 <= KERNL4;

-- Starting Kerning Calculations
when KERNL4 =>
  if (kern < (endRow-startRow)) then
    gRow <= startRow - HighBearingY + charhoriBY + kern;
    ngRow <= startRow - HighBearingY + nextcharhoriBY + kern;
    status_Port1 <= KERNL5;
  else
    status_Port1 <= KERNL11;
  end if;
when KERNL5 =>
  status_Return <= KERNL6;
  RAddress <= RADDRESS + (charindex * maxBheight * 8) + (TO_INTEGER(toUnsigned (charheight,32) all 2)) + (TO_INTEGER(toUnsigned (gRow,32) all 2));
  status_Port1 <= READ0;
when KERNL6 =>
  RK <= TO_INTEGER(unsigned {tempData});
  status_Port1 <= KERNL7;
when KERNL7 =>
  status_Return <= KERNL8;
C. Text Display Engine IP Core

RAAddress <= KADDRESS + (nextcharindex * maxBheight * 8) + (TO_INTEGER(to_unsigned
(ngRow, 32) sll 2));

status_Port1 <= READ0;

when KERNL8 =>

kernSum <= RK + TO_INTEGER(unsigned (tempData));
status_Port1 <= KERNL9;

when KERNL9 =>

kern <= kern + 1;
if (kernSum < Min) then

status_Port1 <= KERNL10;
else

status_Port1 <= KERNL4;
end if;

when KERNL10 =>

Min <= kernSum;
status_Port1 <= KERNL4;

when KERNL11 =>

if (Min > nextcharpitch) then

charadvance <= charpitch - nextcharpitch;
else

charadvance <= charpitch - Min;
end if;

status_Port1 <= KERNL12;

-- Saving kerning data for future use

when KERNL12 =>

midData(0 to 31) <= std_logic_vector (to_unsigned
(charadvance, 32));

status_Port1 <= KERNL13;

when KERNL13 =>

XIL_NPI_Addr_Port1 <= std_logic_vector (to_unsigned ((SKADDRESS +
(TO_INTEGER(to_unsigned(Ll, 32) sll 2)), 32));

XIL_NPI_AddrReq_Port1 <= '1';
XIL_NPI_RNW_Port1 <= '0';
XIL_NPI_Size_port1 <= "0000";
XIL_NPI_RdModWr_Port1 <= '1';
XIL_NPI_WrFIFO_Push_Port1 <= '0';
XIL_NPI_WrFIFO_Flush_Port1 <= '0';
XIL_NPI_WrFIFO_Data_Port1(0 to 7) <= midData(24 to 31);
XIL_NPI_WrFIFO_Data_Port1(8 to 15) <= midData(16 to 23);
XIL_NPI_WrFIFO_Data_Port1(16 to 23) <= midData(8 to 15);
XIL_NPI_WrFIFO_Data.Port1(24 to 31) <= midData(0 to 7);
XIL_NPI_WrFIFO_BE_Port1 <= "0000";
status_Port1 <= KERNL14;

when KERNL14 =>

if XIL_NPI_AddrAck.Port1 = '1' then

XIL_NPI_AddReq_Port1 <= '0';
XIL_NPI_RNW.Port1 <= '1';
XIL_NPI_RdModWr_Port1 <= '0';
XIL_NPI_WrFIFO_Push.Port1 <= '1';
XIL_NPI_WrFIFO_BE.Port1 <= "1111";
status_Port1 <= KERNL15;
end if;

when KERNL15 =>

XIL_NPI_WrFIFO_Push.Port1 <= '0';
status_Port1 <= LAYTS7;

-- End of line and line height calculation

when LAYTS7 =>

LLCounter <= LLCounter + charadvance;
if (charindex = 3) then

WordLength <= 0;
else

WordLength <= WordLength + charadvance;
end if;

if (MaxHeight < charheight) then
C. Text Display Engine IP Core

MaxHeight <= charheight;
end if;
if (MaxBearing < charhoriBY) then
MaxBearing <= charhoriBY;
end if;
if (MaxUnder < chardiff ) then
MaxUnder <= chardiff;
end if;
status_Port1 <= LAYTST;
when LAYTST =>
TLLCounter <= LLCounter + TO_INTEGER(unsigned(CharPropArray(nextcharindex)(0 to 7)));
status_Port1 <= LAYTS8;
when LAYTS8 =>
if ( TLLCounter > AvailableWidth) then
MaxLineHeight <= MaxBearing + MaxUnder;
SMaxLineHeight <= SMaxLineHeight + MaxBearing + MaxUnder;
if (charindex = 3) then
L1 <= p_space;
EndLIndex <= p_space;
LengthIndex <= LengthIndex + AvailableWidth - LLCounter + SpaceLength;
else
if (nextcharindex = 3) then
EndLIndex <= L1 + 1;
LengthIndex <= LengthIndex + AvailableWidth - LLCounter;
LengthIndexS <= TO_INTEGER(unsigned (CharPropArray(3)(24 to 31)));
elsif (nextcharindex /= 3) then
L1 <= p_space;
LengthIndexS <= 0;
EndLIndex <= p_space;
LengthIndex <= LengthIndex + AvailableWidth - LLCounter + WordLength + SpaceLength;
else
end if;
end if;
else
L1 <=L1 + 1;
status_Port1 <= LAYTS9;
end if;
when LAYTS9 =>
middata(0 to 15) <= std_logic_vector (to_unsigned(EndLIndex,16));
middata(16 to 31) <= std_logic_vector (to_unsigned(MaxLineHeight,16));
status_Port1 <= OUTPUT0;
when LAYTS10 =>
if (((SMaxLineHeight > SRow) and (scheck = 0)) then
scheck <= 1;
SLCounter <= LLCounter;
SRBL <= MaxLineHeight - SMaxLineHeight + SRow;
else
if (Alignment = 1) then
StartIndex <= LengthIndex;
elsif (Alignment = 0) then
StartIndex <= 0;
elsif (Alignment = -1) then
StartIndex <= (TO_INTEGER(to_unsigned (LengthIndex,32) srl 1));
else
StartIndex <= LengthIndex;
end if;
MaxBearing <= 0;
MaxUnder <= 0;
MaxHeight <= 0;
status_Port1 <= LAYTS11;
when LAYTS11 =>
middata(0 to 15) <= std_logic_vector(to_unsigned(LengthIndex,16));
middata(16 to 31) <= std_logic_vector(to_unsigned(StartIndex,16));
status_Port1 <= OUTPUT3;
-- Saving line data for future use
when OUTPUT0 =>
  XIL_NPI_Addr_Port1 <= std_logic_vector(to_unsigned((SLADDRESS +
  (TO_INTEGER(to_unsigned(LCounter,32) all 3)),32)));
  XIL_NPI_AddrReq_Port1 <= '1';
  XIL_NPI_rnn_Port1 <= '0';
  XIL_NPI_Size_Port1 <= "0000";
  XIL_NPI_RdModWr_Port1 <= '1';
  XIL_NPI_WrFIFO_Push_Port1 <= '0';
  XIL_NPI_WrFIFO_Flush_Port1 <= '0';
  XIL_NPI_WrFIFO_Data_Port1(0 to 7) <= midData(24 to 31);
  XIL_NPI_WrFIFO_Data_Port1(8 to 15) <= midData(16 to 23);
  XIL_NPI_WrFIFO_Data_Port1(16 to 23) <= midData(8 to 15);
  XIL_NPI_WrFIFO_Data_Port1(24 to 31) <= midData(0 to 7);
  XIL_NPI_WrFIFO_BE_Port1 <= "0000";
  status_Port1 <= OUTPUT1;
when OUTPUT1 =>
  if XIL_NPI_AddrAck_Port1 = '1' then
    XIL_NPI_AddrReq_Port1 <= '0';
    XIL_NPI_rnn_Port1 <= '1';
    XIL_NPI_RdModWr_Port1 <= '0';
    XIL_NPI_WrFIFO_Push_Port1 <= '1';
    XIL_NPI_WrFIFO_BE_Port1 <= "1111";
    status_Port1 <= OUTPUT2;
  end if;
when OUTPUT2 =>
  XIL_NPI_WrFIFO_Push_Port1 <= '0';
  status_Port1 <= LAYTS10;
when OUTPUT3 =>
  XIL_NPI_Addr_Port1 <= std_logic_vector(to_unsigned((SLADDRESS +
  4 + (TO_INTEGER(to_unsigned(LCounter,32) all 3)),32)));
  XIL_NPI_AddrReq_Port1 <= '1';
  XIL_NPI_rnn_Port1 <= '0';
  XIL_NPI_Size_Port1 <= "0000";
  XIL_NPI_RdModWr_Port1 <= '1';
  XIL_NPI_WrFIFO_Push_Port1 <= '0';
  XIL_NPI_WrFIFO_Flush_Port1 <= '0';
  XIL_NPI_WrFIFO_Data_Port1(0 to 7) <= midData(24 to 31);
  XIL_NPI_WrFIFO_Data_Port1(8 to 15) <= midData(16 to 23);
  XIL_NPI_WrFIFO_Data_Port1(16 to 23) <= midData(8 to 15);
  XIL_NPI_WrFIFO_Data_Port1(24 to 31) <= midData(0 to 7);
  XIL_NPI_WrFIFO_BE_Port1 <= "0000";
  status_Port1 <= OUTPUT4;
when OUTPUT4 =>
  if XIL_NPI_AddrAck_Port1 = '1' then
    XIL_NPI_AddrReq_Port1 <= '0';
    XIL_NPI_rnn_Port1 <= '1';
    XIL_NPI_RdModWr_Port1 <= '0';
    XIL_NPI_WrFIFO_Push_Port1 <= '1';
    XIL_NPI_WrFIFO_BE_Port1 <= "1111";
  end if;
when OUTPUT5 =>
  XIL_NPI_WrFIFO_Push_Port1 <= '0';
  LCounter <= 0;
  LCounter <= LCounter + 1;
  LengthIndex <= LengthIndex3;
  L1 <= L1 + 1;
if (LastLine = true) then
    s_Done <= "00000000000000000000000000000001";
    status_Port1 <= INITLC;
else
    status_Port1 <= LAYTS0;
end if;

-- Display Text (Putting characters together in the framebuffer)

when INITLC =>
  if ((ypos + MaxLineHeight) >= BHeight) then
    INITCHECK <= true;
    temph <= 0;
    ypos <= 0;
    CLoop1 <= 0;
    CLoop2 <= 0;
    status_Port1 <= CLEAN0;
  else
    INITCHECK <= false;
    status_Port1 <= INITL0;
  end if;
when INITL0 =>
  INITCHECK <= false;
  if (LCounter = 1) then
    endline <= EndLIndex;
    maxhline <= MaxLineHeight;
    sindexline <= StartIndex;
    status_Port1 <= REPEAT;
  else
    status_Port1 <= INITL1;
  end if;
when INITL1 =>
  RAddress <= SLADDRESS + (TO_INTEGER(to_unsigned (LN, 32) sll 3));
  status_Return <= INITL2;
  status_Port1 <= READ0;
when INITL2 =>
  endline <= TO_INTEGER(unsigned (tempData(0 to 15)));
  maxhline <= TO_INTEGER(unsigned (tempData(16 to 31)));
  status_Port1 <= INITL3;
when INITL3 =>
  RAddress <= SLADDRESS + 4 + (TO_INTEGER(to_unsigned (LN, 32) sll 3));
  status_Return <= INITL4;
  status_Port1 <= READ0;
when INITL4 =>
  sindexline <= TO_INTEGER(unsigned (tempData(16 to 31)));
  if (Alignment = 4) then
    templ <= 0;
  else
    templ <= TO_INTEGER(unsigned (tempData(16 to 31)));
  end if;
  status_Port1 <= REPEAT;
when REPEAT =>
  if (R1 < TextLength) then
    status_Return <= READ0;
    RAddress <= s_indexarray + (TO_INTEGER(to_unsigned (R1, 32) sll 2));
  end if;
  status_Port1 <= READ0;
else
  if (LCounter = 1) then
    cur_ypos <= ypos + maxhline + 1;
  else
    cur_ypos <= ypos + temp + maxhline + 1;
  end if;
  status_Port1 <= DONE;
C. Text Display Engine IP Core

end if;
when READP0 =>
    charindex <= TO_INTEGER(unsigned (tempData));
    status_Port1 <= READA0;
when READA0 =>
    if (charindex = 3) then
        charadvance <= TO_INTEGER(unsigned CharPropArray(charindex)(0 to 7));
    else
        status_Return <= READA1;
        RAddress <= SKADDRESS + (TO_INTEGER(to_unsigned(K1,32) sll 2));
    end if;
end if;
when READA1 =>
    charadvance <= TO_INTEGER(unsigned (tempData));
    status_Port1 <= LBADD0;
when LBADD0 =>
    K1 <= K1 + 1;
    status_Return <= LBADD1;
    RAddress <= s_SAddress + (TO_INTEGER(to_unsigned (charindex,32) sll 2));
end if;
when LBADD1 =>
    SAddress <= TO_INTEGER(unsigned (tempData));
    status_Port1 <= READP1;
when READP1 =>
    charpitch <= TOINTEGER(unsigned (CharPropArray(charindex)(24 to 31)));
    charheight <= TOINTEGER(unsigned (CharPropArray(charindex)(16 to 23)));
    charhoriBY <= TOINTEGER(unsigned (CharPropArray(charindex)(8 to 15)));
    if (R1 = endline) then
        temp <= temp + maxhline + 1;
        status_Return <= READP2;
        RAddress <= SLADDRESS + (TO_INTEGER(to_unsigned ({LN + 1},32) sll 3));
    else
        status_Port1 <= READ0;
    end if;
when READP2 =>
    endline <= TOINTEGER(unsigned (tempData(0 to 15)));
    maxhline <= TOINTEGER(unsigned (tempData(16 to 31)));
    LN <= LN + 1;
    status_Port1 <= READP3;
when READP3 =>
    RAddress <= SLADDRESS + 4 + (TO_INTEGER(to_unsigned (LN,32) sll 3));
end if;
when CLEAN0 =>
    if ((cur_ypos + maxhline) >= BHeight) then
        temp <= 0;
        ypos <= 0;
        CLoop1 <= 0;
        status_Port1 <= CLEAN0;
    else
        status_Port1 <= READP4;
    end if;
when CLEANA =>
    if ((cur_ypos + maxhline) >= BHeight) then
        temp <= 0;
        ypos <= 0;
        CLoop1 <= 0;
        status_Port1 <= CLEANA;
    else
        status_Port1 <= READP5;
    end if;
when CLEANP =>
    if ((cur_ypos + maxhline) >= BHeight) then
        temp <= 0;
        ypos <= 0;
        CLoop1 <= 0;
        status_Port1 <= CLEANP;
    else
        status_Port1 <= READP6;
    end if;
when CLEAN2 =>
    if ((cur_ypos + maxhline) >= BHeight) then
        temp <= 0;
        ypos <= 0;
        CLoop1 <= 0;
        status_Port1 <= CLEAN2;
    else
        status_Port1 <= READP7;
    end if;
when CLEAN1 =>
    if ((cur_ypos + maxhline) >= BHeight) then
        temp <= 0;
        ypos <= 0;
        CLoop1 <= 0;
        status_Port1 <= CLEAN1;
    else
        status_Port1 <= READP8;
    end if;
when CLEAN2 =>
    if ((cur_ypos + maxhline) >= BHeight) then
        temp <= 0;
        ypos <= 0;
        CLoop1 <= 0;
        status_Port1 <= CLEAN2;
    else
        status_Port1 <= READP9;
    end if;
when CLEAN3 =>
    if ((cur_ypos + maxhline) >= BHeight) then
        temp <= 0;
        ypos <= 0;
        CLoop1 <= 0;
        status_Port1 <= CLEAN3;
    else
        status_Port1 <= READP10;
    end if;
when CLEAN4 =>
    if ((cur_ypos + maxhline) >= BHeight) then
        temp <= 0;
        ypos <= 0;
        CLoop1 <= 0;
        status_Port1 <= CLEAN4;
    else
        status_Port1 <= READP11;
    end if;
when CLEAN5 =>
    if ((cur_ypos + maxhline) >= BHeight) then
        temp <= 0;
        ypos <= 0;
        CLoop1 <= 0;
        status_Port1 <= CLEAN5;
    else
        status_Port1 <= READP12;
    end if;
when CLEAN6 =>
    if ((cur_ypos + maxhline) >= BHeight) then
        temp <= 0;
        ypos <= 0;
        CLoop1 <= 0;
        status_Port1 <= CLEAN6;
    else
        status_Port1 <= READP13;
    end if;
when CLEAN7 =>
    if ((cur_ypos + maxhline) >= BHeight) then
        temp <= 0;
        ypos <= 0;
        CLoop1 <= 0;
        status_Port1 <= CLEAN7;
    else
        status_Port1 <= READP14;
    end if;
when CLEAN8 =>
    if ((cur_ypos + maxhline) >= BHeight) then
        temp <= 0;
        ypos <= 0;
        CLoop1 <= 0;
        status_Port1 <= CLEAN8;
    else
        status_Port1 <= READP15;
    end if;
if (CLoop1 < 1920) then
  CLoop2 <= 0;
  status_Port1 <= CLEAN1;
else
  if (INITCHECK = true) then
    status_Port1 <= INITL0;
  else
    status_Port1 <= READP4;
  end if;
end if;

when CLEAN1 =>
  if (CLoop2 < 640) then
    FAddress <= (TO_INTEGER(unsigned (s_FAddress))) +
    (TO_INTEGER(to_unsigned(CLoop1,32) sll 10)) +
    (TO_INTEGER(to_unsigned(CLoop2,32) sll 2));
  status_Port1 <= CLEAN2;
  else
    CLoop1 <= CLoop1 + 1;
    status_Port1 <= CLEAN0;
  end if;
when CLEAN2 =>
  XIL_NPI_Addr_Port1 <= std_logic_vector(to_unsigned(FAddress,32));
  XIL_NPI_AddrReq_Port1 <= '1';
  XIL_NPI_RMW_Port1 <= '0';
  XIL_NPI_Size_Port1 <= "0000";
  XIL_NPI_RdModWr_Port1 <= '1';
  XIL_NPI_WrFIFO_Push_Port1 <= '0';
  XIL_NPI_WrFIFO_Flush_Port1 <= '0';
  XIL_NPI_WrFIFO_Data_Port1(0 to 7) <= "00000000";
  XIL_NPI_WrFIFO_Data_Port1(8 to 15) <= "00000000";
  XIL_NPI_WrFIFO_Data_Port1(16 to 23) <= "00000000";
  XIL_NPI_WrFIFO_Data_Port1(24 to 31) <= "00000000";
  XIL_NPI_WrFIFO_BE_Port1 <= "0000";
  status_Port1 <= CLEAN3;
when CLEAN3 =>
  if XIL_NPI_AddrAck_Port1 = '1' then
    XIL_NPI_AddrReq_Port1 <= '0';
    XIL_NPI_RMW_Port1 <= '1';
    XIL_NPI_RdModWr_Port1 <= '0';
    XIL_NPI_WrFIFO_Push_Port1 <= '1';
    XIL_NPI_WrFIFO_BE_Port1 <= "1111";
    status_Port1 <= CLEAN4;
  end if;
when CLEAN4 =>
  XIL_NPI_WrFIFO_Push_Port1 <= '0';
  CLoop2 <= CLoop2 + 1;
  status_Port1 <= CLEAN1;
when READP4 =>
  sindexline <= TO_INTEGER(unsigned (tempData(16 to 31)));
  if (Alignment = 4) then
    templ <= 0;
    status_Port1 <= CHECKS;
  elsif (Alignment = 2) then
    templ <= 0;
    if (charindex = 3) then
      RI <= RI + 1;
      status_Port1 <= REPEAT;
    else
      status_Port1 <= CHECKS;
    end if;
  else
    templ <= sindexline;
    status_Port1 <= CHECKS;
  end if;
C. Text Display Engine IP Core

```vhdl
when CHECKS =>
  status_Port1 <= READPE;
when READPE =>
  status_Port1 <= START0;
when START0 =>
  if (R2 < charheight) then
    R2MB <= (R2 * maxWidth);
    status_Port1 <= START1;
  else
    R2 <= 0;
    R3 <= 0;
    R1 <= R1 + 1;
    templ <= templ + charadvance;
    status_Port1 <= REPEAT;
  end if;
when START1 =>
  if (R3 < charpitch) then
    FAddress <= \((\text{TO INTEGER}(\text{unsigned}(\text{a_FAddress}))) + \text{TO INTEGER}((\text{to unsigned}(\text{xpos} + \text{temph} + \text{maxline} - \text{charhoriBY} + R2),32) \text{ sll 2})) \text{ sll 10} + \text{to unsigned}(\text{xpos} + \text{templ} + \text{R3}),32) \text{ sll 2});
    RAddress <= SAddress + R2MB + R3;
    status_Return <= WRITE0;
    status_Port1 <= READ0;
  else
    R3 <= 0;
    R2 <= R2 + 1;
    status_Port1 <= START0;
  end if;
when READ0 =>
  XIL_NPI_Addr_Port1 <= std_logic_vector (\text{to unsigned} (RAddress,32));
  XIL_NPI_AddrReq_Port1 <= '1';
  XIL_NPI_RNW_Port1 <= '1';
  XIL_NPI_Size_Port1 <= "0000";
  XIL_NPI_RdFIFO_Pop_Port1 <= '0';
  XIL_NPI_RdFIFO_Flush_Port1 <= '0';
  status_Port1 <= READ1;
when READ1 =>
  if XIL_NPI_AddrAck_Port1 = '1' then
    XIL_NPI_AddrReq_Port1 <= '0';
    XIL_NPI_RNW_Port1 <= '0';
    status_Port1 <= READ2;
  end if;
when READ2 =>
  if XIL_NPI_RdFIFO_Empty_Port1 = '0' then
    XIL_NPI_RdFIFO_Pop_Port1 <= '1';
    if XIL_NPI_RdFIFO_Latency_Port1 = "00" then
      midData(0 to 7) <= XIL_NPI_RdFIFO_Data_Port1(0 to 7);
      midData(8 to 15) <= XIL_NPI_RdFIFO_Data_Port1(8 to 15);
      midData(16 to 23) <= XIL_NPI_RdFIFO_Data_Port1(16 to 23);
      midData(24 to 31) <= XIL_NPI_RdFIFO_Data_Port1(24 to 31);
    end if;
    status_Return <= status_Port1;
  end if;
when READ3 =>
  XIL_NPI_RdFIFO_Pop_Port1 <= '0';
  if XIL_NPI_RdFIFO_Latency_Port1 = "00" then
    status_Return <= status_Port1;
  else
    if XIL_NPI_RdFIFO_Latency_Port1 = "01" then
      midData(0 to 7) <= XIL_NPI_RdFIFO_Data_Port1(0 to 7);
      midData(8 to 15) <= XIL_NPI_RdFIFO_Data_Port1(8 to 15);
      midData(16 to 23) <= XIL_NPI_RdFIFO_Data_Port1(16 to 23);
      midData(24 to 31) <= XIL_NPI_RdFIFO_Data_Port1(24 to 31);
    end if;
    status_Return <= status_Port1;
  end if;
end process;
```
else status_Port1 <= READ4;
end if;
end if;
when READ4 =>
midData(0 to 7) <= XIL_NPI_RdFIFO_Data_Port1(0 to 7);
midData(8 to 15) <= XIL_NPI_RdFIFO_Data_Port1(8 to 15);
midData(16 to 23) <= XIL_NPI_RdFIFO_Data_Port1(16 to 23);
midData(24 to 31) <= XIL_NPI_RdFIFO_Data_Port1(24 to 31);
status_Port1 <= status_Return;

-- Writing bitmap data to frame buffer
when WRITE0 =>
XIL_NPI_Addr_Port1 <= std_logic_vector(to_unsigned(FAddress,32));
XIL_NPI_AddrReq_Port1 <= '1';
XIL_NPI_RNW_Port1 <= '0';
XIL_NPI_Size_Port1 <= "0000";
XIL_NPI_RdModWr_Port1 <= '1';
XIL_NPI_WrFIFO_Push_Port1 <= '0';
XIL_NPI_WrFIFO_BE_Port1 <= "0000";
status_Port1 <= WRITE1;
when WRITE1 =>
if XIL_NPI_AddrAck_Port1 = '1' then
XIL_NPI_AddrReq_Port1 <= '0';
XIL_NPI_RNW_Port1 <= '0';
XIL_NPI_RdModWr_Port1 <= '1';
XIL_NPI_WrFIFO_Push_Port1 <= '1';
R3 <= R3 + 1;
status_Port1 <= WRITE2;
end if;
when WRITE2 =>
XIL_NPI_WrFIFO_Push_Port1 <= '0';
if (R3 < charpitch) then
FAddress <= FAddress + 4;
status_Port1 <= WRITE3;
else status_Port1 <= START1;
end if;
when WRITE3 =>
XIL_NPI_Addr_Port1 <= std_logic_vector(to_unsigned(FAddress,32));
XIL_NPI_AddrReq_Port1 <= '1';
XIL_NPI_RNW_Port1 <= '0';
XIL_NPI_Size_Port1 <= "0000";
XIL_NPI_RdModWr_Port1 <= '1';
XIL_NPI_WrFIFO_Push_Port1 <= '0';
XIL_NPI_WrFIFO_BE_Port1 <= '1';
XIL_NPI_WrFIFO_RdModWr_Port1 <= "0000";
XIL_NPI_WrFIFO_Push_Port1 <= '0';
XIL_NPI_WrFIFO_BE_Port1 <= "0000";
status_Port1 <= WRITE4;
when WRITE4 =>
if XIL_NPI_AddrAck_Port1 = '1' then
XIL_NPI_AddrReq_Port1 <= '0';
XIL_NPI_RNW_Port1 <= '0';
XIL_NPI_RdModWr_Port1 <= '0';
XIL_NPI_WrFIFO_Push_Port1 <= '1';
XIL_NPI_WrFIFO_BE_Port1 <= "1111";
R3 <= R3 + 1;
status_Port1 <= WRITE5;
end if;
when WRITE5 =>
XIL_NPI_WrFIFO_Push_Port1 <= '0';
if (R3 < charpitch) then
  FAddress <= FAddress + 4;
  status_Port1 <= WRITE6;
else
  status_Port1 <= START1;
end if;
when WRITE6 =>
  XIL_NPI_Addr_Port1 <= std_logic_vector(to_unsigned(FAddress,32));
  XIL_NPI_AddrReq_Port1 <= '1';
  XIL_NPI_RNW_Port1 <= '0';
  XIL_NPI_Size_Port1 <= "0000";
  XIL_NPI_RdModWr_Port1 <= '1';
  XIL_NPI_WrFIFO_Push_Port1 <= '0';
  XIL_NPI_WrFIFO_Flush_Port1 <= '0';
  XIL_NPI_WrFIFO_Data_Port1(0 to 7) <= midData(8 to 15);
  XIL_NPI_WrFIFO_Data_Port1(8 to 15) <= midData(0 to 7);
  XIL_NPI_WrFIFO_Data_Port1(16 to 23) <= midData(8 to 15);
  XIL_NPI_WrFIFO_Data_Port1(24 to 31) <= midData(0 to 7);
  XIL_NPI_WrFIFO_BE_Port1 <= "0000";
  status_Port1 <= WRITE7;
end if;
when WRITE7 =>
  if XIL_NPI_AddrAck_Port1 = '1' then
    XIL_NPI_AddrPort1 <= '1';
    XIL_NPI_AddrReq_Port1 <= '0';
    XIL_NPI_RdModWr_Port1 <= '0';
    XIL_NPI_WrFIFO_Push_Port1 <= '1';
    XIL_NPI_WrFIFO_BE_Port1 <= "1111";
    R3 <= R3 + 1;
    status_Port1 <= WRITE8;
  end if;
when WRITE8 =>
  XIL_NPI_WrFIFO_Push_Port1 <= '0';
  if (R3 < charpitch) then
    FAddress <= FAddress + 4;
    status_Port1 <= WRITE9;
  else
    status_Port1 <= START1;
  end if;
when WRITE9 =>
  XIL_NPI_Addr_Port1 <= std_logic_vector(to_unsigned(FAddress,32));
  XIL_NPI_AddrReq_Port1 <= '1';
  XIL_NPI_RNW_Port1 <= '0';
  XIL_NPI_Size_Port1 <= "0000";
  XIL_NPI_RdModWr_Port1 <= '1';
  XIL_NPI_WrFIFO_Push_Port1 <= '0';
  XIL_NPI_WrFIFO_Flush_Port1 <= '0';
  XIL_NPI_WrFIFO_Data_Port1(0 to 7) <= midData(0 to 7);
  XIL_NPI_WrFIFO_Data_Port1(8 to 15) <= midData(0 to 7);
  XIL_NPI_WrFIFO_Data_Port1(16 to 23) <= midData(0 to 7);
  XIL_NPI_WrFIFO_Data_Port1(24 to 31) <= midData(0 to 7);
  XIL_NPI_WrFIFO_BE_Port1 <= "0000";
  status_Port1 <= WRITE10;
end if;
C. Text Display Engine IP Core

XIL_NPI_RNW_Port1 <= '1';
XIL_NPI_RdModWr_Port1 <= '0';
XIL_NPI_WrFIFO_Push_Port1 <= '1';
XIL_NPI_WrFIFO_BE_Port1 <= "1111";
R3 <= R3 + 1;
status_Port1 <= WRITE11;
end if;
when WRITE11 =>
XIL_NPI_WrFIFO_Push_Port1 <= '0';
status_Port1 <= START1;
-- Job is done!
when Done =>
LastLine <= false;
ran_once <= true;
s_Done <= "00000000000000000000000000000010";
status_Port1 <= IDLE;
end case;
end if;
end process displaymem;

tempData(0 to 7) <= midData(24 to 31);
tempData(8 to 15) <= midData(16 to 23);
tempData(16 to 23) <= midData(6 to 15);
tempData(24 to 31) <= midData(0 to 7);
BHeight <= TO_INTEGER(unsigned(s_BHeight));
SRow <= TO_INTEGER(unsigned(s_SRow));
SLine <= TO_INTEGER(unsigned(s_SLine));
AvailableWidth <= TO_INTEGER(unsigned(s_AvailableWidth));
TextLength <= TO_INTEGER(unsigned(s_TextLength));
Alignment <= TO_INTEGER(unsigned(s_Align));
maxBwidth <= (TO_INTEGER(unsigned(s_MaxWidth)));
maxBheight <= (TO_INTEGER(unsigned(s_MaxHeight)));
s_Rst <= slv_reg0(31);
s_E <= slv_reg1(31);
s_xpos <= slv_reg2;
s_ypos <= slv_reg3;
s_AvailableWidth <= slv_reg4;
s_TextLength <= slv_reg5;
s_MaxWidth <= slv_reg6;
s_MaxHeight <= slv_reg7;
s_FAddress <= slv_reg12;
s_Align <= slv_reg13;
s_SLine <= slv_reg15;
s_SRow <= slv_reg16;
s_BHeight <= slv_reg17;
slv_reg8 <= std_logic_vector(to_unsigned(cur_ypos, 32));
slv_reg11 <= CYCLES(31 downto 0);
slv_reg19 <= s_Done;
end IMP;
The sample API code presented in this section is intended to demonstrate the performance of the designed engine in a simple manner. This is the not an API used by WebKit when using the designed engine. For WebKit to use the engine, some changes have been made in the source code of WebKit in the RenderBlockLineLayout.cpp file.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <time.h>
#include <memory.h>

//***************Added for fb0 usage***************
#include <linux/fb.h>
//***************End of Added for fb0 usage***************

//********** Added for Hardware and Memory Control **********/
define DRIVER "/dev/mem"
// register of sets
#define R_RST                 0
#define R_E                   1
#define R_XPOS                2
#define R_YPOS                3
#define R_AvailableWidth      4
#define R_TextLength          5
#define R_MaxWidth            6
#define R_MaxHeight           7
#define R_endLarray           8
#define R_BAddress            11
#define R_FAAddress           12
#define R_ALIGNMENT           13
#define R_SLine               15
#define R_SRow                16
#define R_BHeight             17
#define R_DONE                19

#define INT2BIN(x) { \ int i; \
 if((x & (1<<31)) == 2147483648) \ printf("1");
```
else\n   printf("0");\nfor(i=30;i>=0;i--)\n   { printf("%u", (x & (1<<i))>>i ); \n     if (i%4==0) \n       printf(" "); \n   printf("n"); } \n
//generics
#define MAP_SIZE 32768UL// 32k (8 pages size)
#define K64_SIZE 65536UL// 64k (16 pages size)
#define BIG_SIZE 262144UL// 256K (64 pages size)
#define GNT_SIZE 1048576UL// 1M (256 pages size)
#define MAP_MASK (MAP_SIZE - 1)
#define K64_MASK (K64_SIZE - 1)
#define BIG_MASK (BIG_SIZE - 1)
#define GNT_MASK (GNT_SIZE - 1)

/****************** Added for FreeType *******************/
#include <ft2build.h>
#include FT_FREETYPE_H
#include FT_GLYPH_H
#include FT_OUTLINE_H
#define CHARSIZE    400 /* character point size */
#define MAX_GLYPHS 5120 /* Maximum number of glyphs rendered at one time */

FT_Error error;
FT_Library library;
FT_UInt gindex;
FT_Face face;
FT_Glyph glyphs[MAX_Glyphs];

/*********************** Global Variables **********************/
int LCounter = 0, Lines = 0;
int *LengthIndex, JustAdd, *SpaceCount;
int FAddress;
int SumWidth = 0;
int AvrWidth = 0;

int num_glyphs;
int tab_glyphs;
int cur_glyph;
int Fail;
int Num;
short antialias = 0; /* smooth fonts with gray levels */
short force_low;

int fd;
unsigned int *mem_map_base_1;
unsigned char *mem_map_base_2;
unsigned char *mem_map_base_5;
unsigned int *mem_map_base_6;
unsigned int *mem_map_base_7;
unsigned long *reg_map_base;
unsigned int *IndexArray_VA;
unsigned char *CharPropArray_VA;
unsigned char *Bitmap_VA;
unsigned int *SAddress_VA;
unsigned int *Kerning_VA;
unsigned int *Reg_addr_VA;

unsigned int mem_physaddr_1 = 0x5E700000; /* Character Index Array*/
unsigned int mem_physaddr_2 = 0x5E710000; /* Character Properties*/
unsigned int mem_physaddr_5 = 0x5E800000; /* Glyph Bitmaps */
unsigned int mem_physaddr_6 = 0x5E780000; /* Glyph Bitmaps Addr. */
unsigned int mem_physaddr_7 = 0x5E900000; /* Glyphs Kerning Data */
unsigned int reg_physaddr = 0xB6E00000; /* Control Registers */

volatile unsigned long Cycles1, Cycles2, Cycles3, Cycles4, Cycles5;
unsigned long long TotalCycles1 = 0, TotalCycles2 = 0, TotalCycles3 = 0;

volatile unsigned long *cur_ypos;

/******************** Text Structures ********************/

struct Characters_Data
{
    unsigned char *CharBitmap;
    int CharHeight;
    int CharHorBearingY;
    int CharAdvance;
    int CharPitch;
    int *RK;
    int *LK;
} My_Character[100];

struct Text_Data
{
    int *IndexArray;
    int AvailableWidth;
    int TextLength;
    int xpos;
    int ypos;
    int bheight;
    int srow;
    int maximumH;
    int maximumW;
    int SLCounter;
    int SRBL;
    int LCounter;
} My_Text;

struct Line_Data
{
    int *StartIndex;
    int *MaxLineHeight;
    int *EndLIndex;
    int LCounter;
    int Lines;
} My_Line;

int pixel_size = 18, Alignment = 2;

char * text;

/******************** Function Declaration ********************/
D. Sample API Code

```c
int file_setup();
void file_cleanup();
int HaltungControl();
int ParagraphCalculations(int Alignment);
int FreetypeDisplay();

static void Panic(const char* message)
{
    fprintf(stderr, "%sn", message);
    exit(1);
}

static void Usage(void)
{
    fprintf(stderr, "Simple test script for the FreeType based text rendering
engine "n");
    fprintf(stderr, "-------------------------------------------------------------------"n");
    fprintf(stderr, "Usage: fttimer [options] fontname.ttf|ttc\n"n");
    fprintf(stderr, "options:
"n");
    fprintf(stderr, "-w : Available Width (default is 640)\n"n");
    fprintf(stderr, "-s : character pixel size (default is 24)\n"n");
    fprintf(stderr, "-l : Number of glyphs to be rendered and displayed (default is
1230)\n"n");
    fprintf(stderr, "-a : Alignment (1:Right,2:Left,3:Center,4:Justify (default is
4)\n"n");
    fprintf(stderr, "-x : x position of the text box on screen (default is 0)\n"n");
    fprintf(stderr, "-y : y position of the text box on screen (default is 0)\n"n");
    exit(1);
}

/**********************************************************/
/*                                                                 */
/*  Get_Time:                                                      */
/*                                                                 */
/*    Returns the current time in milliseconds.                    */
/*                                                                 */
/**********************************************************/

static long Get_Time(void)
{
    return clock() * 1000 / CLOCKS_PER_SEC;
}

/**********************************************************/
/*                                                                 */
/*  LoadChar:                                                      */
/*                                                                 */
/*    Loads a glyph into memory.                                   */
/*                                                                 */
/**********************************************************/

FT_Error LoadChar(int idx, FT_Face face)
{
    FT_Glyph glyph = *(Reg_addr_VA + R_BAddress);

    /* load the glyph in the glyph slot */
    error = FT_Load_Glyph(face, idx, FT_LOAD_DEFAULT) ||
            FT_Get_Glyph(face->glyph, &glyph);
    if (!error)
    {
        
        144
```
D. Sample API Code

```c
My_Character[idx].CharHorBearingY = (face->glyph->metrics.horiBearingY) >> 6;
*({unsigned char *)(CharPropArray_VA + idx*4 + 2}) = face->glyph->
  >metrics.horiBearingY >> 6;

My_Character[idx].CharAdvance = (face->glyph->metrics.horiAdvance) >> 6;
*({unsigned char *)(CharPropArray_VA + idx*4 + 3}) = face->glyph->
  >metrics.horiAdvance >> 6;

if (idx == 3) My_Character[idx].CharPitch = (face->glyph->metrics.horiAdvance) >> 6;
if ((face->glyph->metrics.horiBearingY & 63) != 0) printf("Hey! They aren't
  zero!\n");
glyphs[idx] = glyph;
}
return error;
}

FT_Error ConvertRaster( int idx )
{
  FT_Glyph bitmap;
  FT_BitmapGlyph glyph_bitmap;
  int j;
  bitmap = glyphs[idx];
  if (bitmap->format == FT_GLYPH_FORMAT_BITMAP )
    error = 0; /* we already have a (embedded) bitmap */
  else
  {
    error = FT_Glyph_To_Bitmap( &bitmap,
                                  antialias ? FT_RENDER_MODE_NORMAL :
                                            FT_RENDER_MODE_MONO,
                                  0,
                                  0);

    glyph_bitmap = (FT_BitmapGlyph)bitmap;

    My_Character[idx].CharHeight = glyph_bitmap->bitmap.rows;
    *((unsigned char *)(CharPropArray_VA + idx*4 + 1)) = glyph_bitmap->bitmap.rows;

    if (idx != 3)
      {
        My_Character[idx].CharPitch = glyph_bitmap->bitmap.pitch;
        *((unsigned char *)(CharPropArray_VA + idx*4 + 2)) = glyph_bitmap->bitmap.pitch;
      }

    if (My_Text.maximumH < My_Character[idx].CharHeight) My_Text.maximumH =
      My_Character[idx].CharHeight;
    if (My_Text.maximumW < My_Character[idx].CharPitch) My_Text.maximumW =
      My_Character[idx].CharPitch;
    SumWidth = SumWidth + My_Character[idx].CharAdvance;
    if (glyph_bitmap->bitmap.rows)
```

145
D. Sample API Code

```c
{  
  My_Character[ idx ].CharBitmap = ( unsigned char* ) malloc( glyph_bitmap->bitmap.rows*glyph_bitmap->bitmap.pitch );
  My_Character[ idx ].LK = ( int* ) malloc( glyph_bitmap->bitmap.rows*sizeof( int ) );
  My_Character[ idx ].RK = ( int* ) malloc( glyph_bitmap->bitmap.rows*sizeof( int ) );

  for ( j = glyph_bitmap->bitmap.rows - 1; j >= 0; j-- )
  {
    memcpy( & ( My_Character[ idx ].CharBitmap[ j*glyph_bitmap->bitmap.pitch ] ),
             & ( glyph_bitmap->bitmap.buffer[ j*glyph_bitmap->bitmap.pitch ] ), glyph_bitmap->bitmap.pitch );
  }
}
```

```c
}  
else  
{  
  My_Character[ idx ].CharBitmap = 0;
  }

  if ( !error )
  {  
    FT_Done_Glyph( bitmap );
  }
}

  return error;
}

int main( int argc, char** argv )
{
  int i, j, k, p, pix;
  char  fontname[ 128 + 4 ];
  char*  execname;
  FT_UInt  gindex;

  execname = argv[ 0 ];
  My_Text.AvailableWidth = 640;
  My_Text.xpos = 0;
  My_Text.ypos = 0;
  My_Text.srow = 0;
  My_Text.bheight = 480;
  My_Text.SRBL = 0;
  My_Text.SLCounter = 0;

  while ( argc > 1 && argv[ 1 ][ 0 ] == ' ' )
  {
    switch ( argv[ 1 ][ 1 ] )
    {
      case 'w':
        argc--;  
        argv++;
        if ( argc < 2 || sscanf( argv[ 1 ], "%d", &My_Text.AvailableWidth ) != 1 )
          Usage();
        break;
      case 'l':
        argc--;  
        argv++;
        if ( argc < 2 || sscanf( argv[ 1 ], "%d", &My_Text.TextLength ) != 1 )
```
D. Sample API Code

Usage();
break;

case 'a':
    argc--;
    argv++;
    if ( argc < 2 ||
         sscanf( argv[1], "%d", &Alignment ) != 1 )
        Usage();
    break;

case 's':
    argc--;
    argv++;
    if ( argc < 2 ||
         sscanf( argv[1], "%d", &pixel_size ) != 1 )
        Usage();
    break;

case 'x':
    argc--;
    argv++;
    if ( argc < 2 ||
         sscanf( argv[1], "%d", &My_Text.xpos ) != 1 )
        Usage();
    break;

case 'y':
    argc--;
    argv++;
    if ( argc < 2 ||
         sscanf( argv[1], "%d", &My_Text.ypos ) != 1 )
        Usage();
    break;

case 'f':
    argc--;
    argv++;
    if ( argc < 2 ||
         sscanf( argv[1], "%d", &FAddress ) != 1 )
        Usage();
    break;

case 'b':
    argc--;
    argv++;
    if ( argc < 2 ||
         sscanf( argv[1], "%d", &My_Text.srow ) != 1 )
        Usage();
    break;

case 'h':
    argc--;
    argv++;
    if ( argc < 2 ||
         sscanf( argv[1], "%d", &My_Text.bheight ) != 1 )
        Usage();
    break;

default:
    fprintf( stderr, "Unknown argument \"%s\"\n", argv[1] );
Usage();
break;
}
D. Sample API Code

```c
argc--;
argv++;
}

if ( argc != 2 )
    Usage();

i = strlen( argv[1] );
while ( i > 0 && argv[1][i] != '\\' )
{
    if ( argv[1][i] == '.' )
    {
        i = 0;
        i--;
    }
}

fontname[128] = '\0';
strncpy( fontname, argv[1], 128 );

if ( i >= 0 )
{
    strncpy( fontname + strlen( fontname ), "\.ttf", 4 );
}

if ( file_setup() <0 )
    return -1;

/* Initialize engine */

if ( ( error = FT_Init_FreeType( &library ) ) != 0 )
    Panic( "Error while initializing engine" );

error = FT_New_Face( library, fontname, 0, &face );
if ( error == FT_Err_Cannot_Open_Stream )
    Panic( "Could not find/open font resource" );
else if ( error )
    Panic( "Error while opening font resource" );

int total,rendered_glyphs;
antialias = 1;
force_low = 0;

/* get face properties and allocate preload arrays */

num_glyphs = face->num_glyphs;

tab_glyphs = MAX_GLYPHS;
if ( tab_glyphs > num_glyphs )
    tab_glyphs = num_glyphs;

error = FT_Set_Pixel_Sizes( face, pixel_size, pixel_size );
if ( error )
    Panic( "Could not reset instance" );

Num = 0;
Fail = 0;

total = num_glyphs;
rendered_glyphs = 0;

cur_glyph = 0;

Cycles4 = *(unsigned long *)(Reg_addr_VA + R_BAddress);
```
D. Sample API Code

for ( Num = 0; Num < 94; Num++ )
{
    error = LoadChar( Num, face );
    if ( error )
    {
        Fail++;
        total--;
    }
}

for ( Num = 0; Num < 94; Num++ )
{
    if ( ( error = ConvertRaster( Num ) ) != 0 )
        Fail++;
    else
        rendered_glyphs++;
}

/* Now free all loaded outlines */
for ( Num = 0; Num < 94; Num++ ) FT_Done_Glyph( glyphs[Num] );

Cycles5 = *(unsigned long *)(Reg_addr VA + R_BAddress);
if (Cycles5 < Cycles4) TotalCycles3 = TotalCycles3 + ((unsigned long)(0xFFFFF4FE) - Cycles4) + Cycles5;
else TotalCycles3 = TotalCycles3 + (Cycles5 - Cycles4);

/* Process the Text input file */
i = 0;
j = 0;
char *pptc;
int text_size[10000];
FILE *input1;
int text_size_prv = 0, text_size_cur = 0, totalchars = 0, totalp = 0;
pptc = (char*) malloc (sizeof(char)*2);
input1 = fopen("input.txt", "r");
printf("Processing Input Text File...
");
while(pptc[0]!=EOF)
{
    pptc[0]=fgetc(input1);
totalchars++;
    if (pptc[0]=='
')
    {
        pptc[0]=fgetc(input1);
        if (pptc[0]=='
')
        {
            fseek(input1, 0, SEEK_CUR);
text_size_cur = ftell(input1);
text_size[i] = text_size_cur - text_size_prv - 2;
totalchars = totalchars - 2;
i++;
text_size_prv = text_size_cur;
        }
    }
}

fseek(input1, 0, SEEK_SET);
totalp = i;

*(Reg_addr VA + R_YPOS) = My_Text.ypos;

/* Prepare Data in RAM for Hardware Access */
D. Sample API Code

```c
if ( My_Text.maximumW % 4 != 0 )
    My_Text.maximumW = My_Text.maximumW + 4 -
        (My_Text.maximumW % 4);

for (i=0; i<94; i++)
{
    *(unsigned int *) (SAddress_VA + i) =
        1585446912 + (i * My_Text.maximumH *
        My_Text.maximumW * 2);

    for (j=0; j<My_Character[i].CharHeight; j++)
    {
        for (k=0; k<My_Character[i].CharPitch; k++)
        {
            *(unsigned char *) (Bitmap_VA +
                (i * My_Text.maximumH * My_Text.maximumW * 2) +
                (j*My_Text.maximumW+k)) = My_Character[i].CharBitmap[
                ((i*My_Character[i].CharPitch+k)];
        }
    }
}

for (i=0; i<94; i++)
{
    for (j=My_Character[i].CharHeight-1; j>=0; j--)
    {
        My_Character[i].LK[j]=0;
        for (pix = 0; pix < My_Character[i].CharPitch; pix++)
        {
            if ( My_Character[i].CharBitmap[j*My_Character[i].CharPitch + pix] == 0 )
            My_Character[i].LK[j]++;
            else break;
        }
    }
}

for (i=0; i<94; i++)
{
    for (j=My_Character[i].CharHeight-1; j>=0; j--)
    {
        My_Character[i].RK[j]=0;
        for (pix = My_Character[i].CharPitch-1; pix >=0; pix--)
        {
            if ( My_Character[i].CharBitmap[j*My_Character[i].CharPitch + pix] == 0 )
            My_Character[i].RK[j]++;
            else break;
        }
    }
}

for (j=0; j<My_Character[3].CharHeight; i++)
{
    *(unsigned int *) (Kerning_VA +
        (i * My_Text.maximumH * 2) + j) =
        My_Character[3].CharHeight + j;
    printf("RK=\%d, \n", *(unsigned int *) (Kerning_VA +
        (i * My_Text.maximumH * 2) +
        My_Character[3].CharHeight + j));
}

printf("\n");
}

for (j=0; j<My_Character[3].CharHeight; i++)
{
    *(unsigned int *) (Kerning_VA +
        (3 * (My_Text.maximumH * 2)) +
        My_Character[3].CharHeight + j) = 0;
    *(unsigned int *) (Kerning_VA +
        (3 * (My_Text.maximumH * 2)) + j) = 0;
}

/*Processing Text in the Hardware (Layout and Display) */
for(j=0; j<totalp; j++)
{
```
cur_ypos = (unsigned long *)(Reg_addr_VA + R_endLarray);
text = (char*) malloc(sizeof(char)*text_size[j]);
fwrite(text, 1, text_size[j], input1);
My_Text.TextLength = text_size[j];
for (p=0; p<My_Text.TextLength; p++)
{
    gindex = FT_Get_Char_Index(face, text[p]);
    if (gindex==0) *(((unsigned int *)(IndexArray_VA + p)) = 3;
    else *((unsigned int *)(IndexArray_VA + p)) = gindex;
}

HardwareControl();
if (*cur_ypos>=480) *(Reg_addr_VA + R_YPOS) = 480;
else *(Reg_addr_VA + R_YPOS) = *cur_ypos + My_Text.maximumH;
free(text);

file_cleanup();
FT_Done_Face(face);
FT_Done_FreeType(library);
return 0;

/**************************************************************/
/*This function opens the /dev/mem file, and mmap the memory!*/
/**************************************************************/

int file_setup()
{
    // open the dev file
    fd = open("/dev/mem",O_RDWR | O_SYNC);
    if (fd<0)
    {
        printf("Failure to open the file\n");
        return -1;
    }
    else
    {
        reg_map_base = (unsigned long *)mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, reg_physaddr & ~MAP_MASK);
        if(reg_map_base < 0)
            goto fail_mmap2;
        Reg_addr_VA = reg_map_base + (reg_physaddr & MAP_MASK);

        mem_map_base_1 = (unsigned int *)mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, mem_physaddr_1 & ~MAP_MASK);
        if(mem_map_base_1 < 0)
            goto fail_mmap1;
        IndexArray_VA = mem_map_base_1 + (mem_physaddr_1 & MAP_MASK);

        mem_map_base_2 = (unsigned char *)mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, mem_physaddr_2 & ~MAP_MASK);
        if(mem_map_base_2 < 0)
            goto fail_mmap1;
D. Sample API Code

CharPropArray_VA = mem_map_base_2 + (mem_physaddr_2 & MAP_MASK);
mem_map_base_5 = (unsigned char *)mmap(0, GNT_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, mem_physaddr_5 & ~GNT_MASK);
if(mem_map_base_5 < 0)
goto fail_mmap1;
Bitmap_VA = mem_map_base_5 + (mem_physaddr_5 & GNT_MASK);
mem_map_base_6 = (unsigned int *)mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, mem_physaddr_6 & ~MAP_MASK);
if(mem_map_base_6 < 0)
goto fail_mmap1;
SAddress_VA = mem_map_base_6 + (mem_physaddr_6 & MAP_MASK);
mem_map_base_7 = (unsigned int *)mmap(0, K64_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, mem_physaddr_7 & ~K64_MASK);
if(mem_map_base_7 < 0)
goto fail_mmap1;
Kerning_VA = mem_map_base_7 + (mem_physaddr_7 & K64_MASK);
return 0;

fail_mmap2:
    printf("Failure to mmap registers\n");

fail_mmap1:
    printf("Failure to mmap memory\n");
    close(fd);
    return -1;

}
D. Sample API Code

```c
*(Reg_addr_VA + R_Alignment) = Alignment;
*(Reg_addr_VA + R_SLLine) = My_Text.SLCounter;
*(Reg_addr_VA + R_SRow) = My_Text.SRBL;
*(Reg_addr_VA + R_BHeight) = My_Text.bheight;
*(Reg_addr_VA + R_E) = 1;

do{
    CHKDone = (unsigned long *)(Reg_addr_VA + R_DONE);
}
}while(*CHKDone < 1);

Cycles2 = *(unsigned long *)(Reg_addr_VA + R_BAddress);
if (Cycles2 < Cycles1) TotalCycles1 = TotalCycles1 + ((unsigned long)(0xFFFFFFFF) - Cycles1) + Cycles2;
else TotalCycles1 = TotalCycles1 + (Cycles2 - Cycles1);

do{
    CHKDone = (unsigned long *)(Reg_addr_VA + R_DONE);
}
}while(*CHKDone < 2);

Cycles3 = *(unsigned long *)(Reg_addr_VA + R_BAddress);
if (Cycles3 < Cycles1) TotalCycles2 = TotalCycles2 + ((unsigned long)(0xFFFFFFFF) - Cycles1) + Cycles3;
else TotalCycles2 = TotalCycles2 + (Cycles3 - Cycles1);

*(Reg_addr_VA + R_E) = 0;
return 0;
```
Vita Auctoris

Soheil Servati Beiragh was born in 1982, in Tehran, IRAN. He received his B.A.Sc. in 2004 from Electrical and Computer Engineering Department of University of Tehran and M.Sc. in 2008 from Electrical Engineering Department of Amirkabir University of Technology (Tehran Polytechnic) two of the most prestigious universities in IRAN. His research interest includes VLSI and Embedded System Design, Computer Architecture, Mobile Computing and GPU acceleration.