Digital Slate: Replacing the Paper Notepad

Kumar Anik
Student, IIIT Allahabad, B-Tech(IT)
703-BH3, Indian Institute of Information Technology
Allahabad, Deoghat Jhalwa , UP-211012

Abhishek Sharma
Student, IIIT Allahabad, B-Tech(IT)
719-BH3, Indian Institute of Information Technology
Allahabad, Deoghat Jhalwa , UP-211012

ABSTRACT
"Save paper, save trees" is a hot agenda in today's world. This paper presents an ingenuous approach to achieve the motto, by proposing a software-only design, through modeling a paper notepad digitally. It describes the financial and social significance of this model underlining its implementation. The paper also briefs about the underlying hardware selected for the design implementation. The hardware utilizes ARM processor and is quite portable. Its known as Mini2440 FriendlyARM. Throughout the paper, optimization (power, memory etc.) has been primarily focused yielding to a recommendable solution.

Categories and Subject Descriptors
C.3 [**Special Purpose and Application based Systems**]: Real-time and embedded systems.

General Terms
Management, Measurement, Performance, Design, Economics, Experimentation, Human Factors.

Keywords
Digital Slate, power-efficient design, low power, embedded device.

1. INTRODUCTION
As embedded systems, currently, are moving towards versatility, their cost has increased by a substantial amount. Though their current cost is lesser than their original inventive cost due to the adaptations in the field of technology, yet finding a cheap device dedicated to a specific task is a tough job. The normal consumer electronics shall only comprise of high-end ultra-versatile devices which are multi-solution focused and hence cost greater as per the required usage. This paper is dedicated to one such task specific scenario. Paper notepads are a product which cover a wide range of paper market including graph-pads, copies, books etc. The device proposed further has in its design the core idea to "save trees" by replacing the conventional "paper" notepads. For this particular implementation a specific device namely MINI2440 Friendly ARM[6] has been considered, on which a minimalistic environment similar to a paper notepad is generated. It has been kept in mind all the way long from design to implementation of the environment to be as less resource consuming as possible, to facilitate scaling down of hardware and thus making it cheaper. All the work proposed here is based entirely on the software design rather than focusing on internals of hardware structure.

1.1 Prior Work
For this desired result, current technologies offer a few options

1. A normal laptop or the latest coming up of the netbooks, could double up for the purpose. So along with their usual stuff, they would have been equipped with a basic application which provides the usual note-making facilities. Separate input devices can always be attached to provide a interface similar to the common paper notepad. But the problem would arise owing to its inherited bulkiness, and to most of the paper notepad users, it wont remain affordable.
2. High end mobile phones and PDAs can also provide an alternative, but problems is more or less the same as with laptops, namely their costs, and reduced battery life owing to their other provisional services.

1.2 Paper Schema
This paper intends to describe the merging of various appropriate technologies to replicate the environment of a paper notepad. The appropriateness of these technologies is stated in the terms of the design proposed. Furthermore, the paper demonstrates a few techniques specific to the environment which introduce further improvement in power-efficiency.
Section 2 describes some high level designs which may have been speculated considering the common embedded software technologies. Next it focuses on an implementation specific design which serves "better" for the primary objective.
Section 3 presents the significance of such a device, with a social and financial perspective.
Section 4 includes a brief procedure to realize the design. Section 5 depicts a few experimental results supporting the idea.,

2. DESIGN OVERVIEW
To achieve a digital environment that can act as a replica of a paper notepad we need a hardware which can be of a comparable physical size of a notepad and supports handwriting as input and displays the same. So basically we are asking for a touch screen and a board which has a decent processor so as to compute all the necessary details of the environment. For this we can choose any hand-held complying to our specifications. After that we try to figure out what sorts of software design can recreate the maximum possible genuine notepad environment and yet consume the minimum available resources.

2.1 Obvious Designs
Using day-to-day software traditions and designs, to replicate a notepad some ingenuous designs strike us right at the beginning. While starting the development of an embedded

application misnomers like J2ME and symbian (for mobile phones) are often encountered. For java development J2ME can also be used. So for developing any embedded application, we need an OS that supports embedded systems. There are quite a many existing at present namely, WinCE, Angstrom, OPIE, Qtopia, GPE and many more. Along with the OS, the respective app design changes accordingly. Besides, since our objective of a notepad environment requires a UI for interaction with user, the chosen OS must naturally support GUI rendering.

So we first discuss more popularly used approaches in embedded development.

1. Any embedded OS can be used with J2ME sitting over it and so a nice Java MIDlet, which is our notepad app, can be developed.
2. U

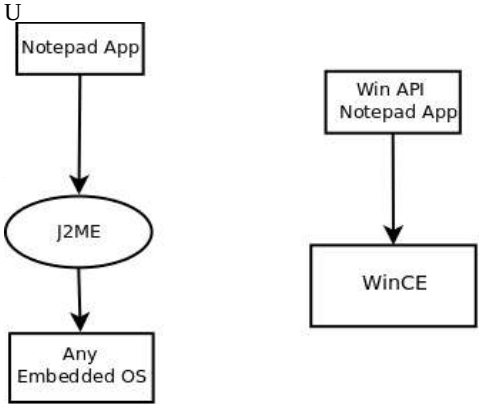

Illustration 1: OS environment

w
indow API based app. can be developed.

After taking these designs into consideration, we found that there can be better approaches as in the case that we can go for solutions without any middle layer (as java) between the app and the OS, and/or we may also wish to make changes to the kernel and so *nix based OS can prove to be much helpful[5]. In the former case performance can be increased by reducing layers between app and OS. As to the issue on changing the kernel behavior we might wish to change the kernel interrupt handling procedure, some functionalities etc. Catering to these improvements we move further to develop more designs.

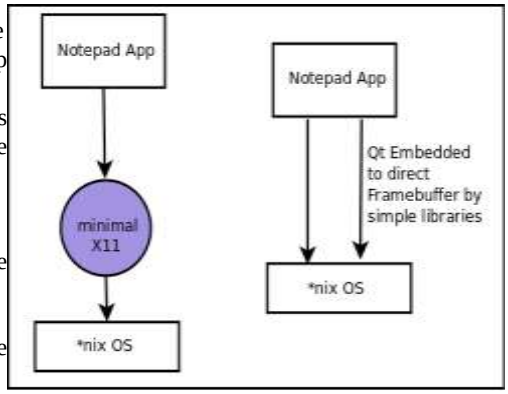

Illustration 2: App Designs

Coming to FOSS [Free and Open Source Software] and non-JAVA based development, it is easy to tell that all we need is a linux distro which can run efficiently on the device. While in the case of WinCE, development should be related to the OS architecture, it's not the case when considering *nix. Since we wish to develop an app which user can "see", then what we actually require is not the details of an OS but of the windowing system for the OS. The app is supposed to communicate to the windowing system for resource specific details rather than direct communication with OS. It is here the revelation took place that for embedded devices there exist two possible windowing solutions for deploying GUI apps.

1. X11 – A widely used windowing system for *nix with server capabilities of porting GUI remotely. Utilizes a protocol stack to render GUI.
2. Qt Embedded Environment(QTE) – Deviates from X11 in terms of server capabilities and GUI rendering, since it doesn't support network UI porting and renders directly to the frame buffer.

But again there "seems" to be one layer sitting in between(although not the case as in Qt Embedded) the app and the OS. But since everything is FOSS now and since there is no need for a VM to be strictly sitting for development, we can actually go for modifying the source of X11(not QTE) so as to increase performance.

2.2 Proposed Design

After the consideration of the above designs a final derivation

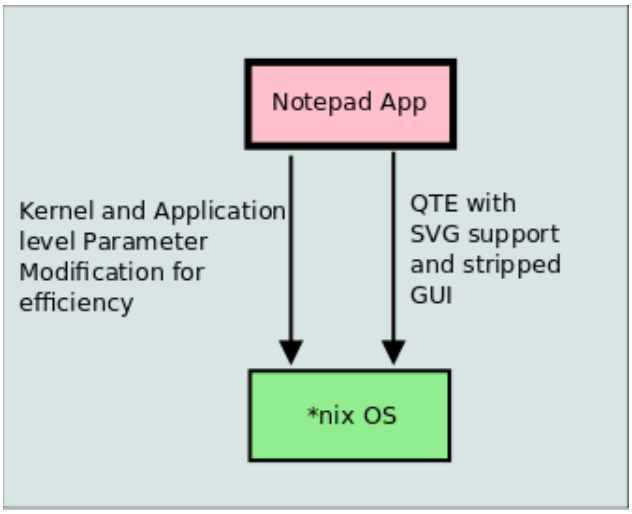

Illustration 3: Final implementation

is proposed, matching our requirements. We choose the QTE based solution, where we do not keep any desktop environment and make our app the sole GUI on the device. Since QTE is not a layer of windowing system, the desktop in QTE based distros is nothing more than a pluggable module. All other apps which are displayed on the desktop are actually independent of it, and could be run on the device even without it. Apart from this

"load" removal, QTE has another advantage that it supports SVG[13] graphics inherently. Since we are developing a slate environment we only need a few shades of black (gray-scale) to allow user to write/draw. It is proven that for less colors (our requirements are even lesser) an SVG image takes up much fraction of storage area eaten by raster graphics. Therefore with these inherent properties of QTE we have actually finalized a simple and yet efficient design.

2.3 Comparison

With the design finalized we need to sort out the advantages of it over the others. Building up the significance right from the bottom, we chose *nix as the heart of the device. Since these OS'es are FOSS, implementation and requirement based modification can be constructed on the kernel itself. Some techniques discussed later in Section 4 can be implemented on the kernel, and hence the need for *nix. Coming back to previous designs most of them required a middle layer to handle the application level params. Therefore utilizing QTE in the design subtracts one extra layer process to keep running for monitoring apps. With QTE's buffer draw mechanism, application can directly ask the OS to give it the resources as contrasted to the indirect middle layer approach. Developing in QTE also provides two extra vantage points. First that QT itself follow object-oriented approach(OOA) through c++ and encourages the same in terms of application development. Without getting into further details, it is to be stated that implementation itself gets better using OOA. Second that QTE is specifically designed for efficient code development when it comes to less resource consumption. Direct Buffer Rendering is a proof for that cause and SVG support is a by-product. SVG unlike raster graphics can zoom upto 800 %(even more) and occupies much lesser storage. Since SVG is a collection of xml files, a simple "gz" compression can reduce its size to 20% of original. Thus both memory and processing requirements are minimized in the design.

3. SIGNIFICANCE

As described in Section 1, the DIGITAL SLATE focuses on specialization rather than versatility. Specialization drives to compactness of the model and less resource consumption in our case. Both these factors indirectly impact the usage and implementation of the device which in turn can act as a catalyst for global and economical factors.

3.1 Global Impact

In U.S., on an average a person uses 700 pounds of paper annually[9]. If only a single run of Sunday New York Times is recycled 75000 trees could be saved[10]. The point to be stated is that, we use paper a lot and environment is harmed as a consequence. The device and the original idea were originated in order to save the Green.

For further illustration, lets take a scenario of a school where kids are supposed to maintain paper notebooks for note-making etc. Now there are normally about 7-10 subjects in a year and they might demand 1-2 or more notepads for each. So on average a student will use at the minimum 10 copies. All statistics are at the minimum(not average). Furthermore the average number of students per school in the entire country, using 10 notepads each, is 89599(as taken in year 1999-2000)[11]. So per school, we have a consumption of 895990 notebooks and there are many schools in US as we know; naturally, trees get squandered. Therefore a device which can replace notepads providing similar usage will do the trick. Although one may have gone for a "richer" multimedia solution, but why go for that when the above stated device can be much cheaper and robust. Thus the device can be as green and pocket friendly as possible.

3.2 Financial Perspective

Average cost of a legal lined paper notepad is about 2$. Referring to the above number of notepads used per school, we have 895990*2 $ of money per school. Per head we have 20$ a year on paper notepads. This 20$ sums up to 200$ in 10 years. The device which we have specifically used in order to create a notepad environment(MINI2440), costs lesser than 100$. It can be used to replace all notebooks of different subject in one shot(digital memory can store more than a paper). So if maintained properly, Digital Slate can serve a person for entire lifetime eliminating scope of paper consumption. In terms of power usage Mini2440 consumes 5V and doesn't have any batteries installed but that is achievable simply by using 4 NiMH rechargeable batteries. The design implementation, discussed in the section 4, focuses on less power utilization as that is also an impact factor to the cost incurred.

3.3 Specific to the device and app.

Until and unless we write on a notepad with a pencil, we can safely assert that the copy is not dynamic in nature. Once written always remains. In such a case managing two or more subjects is difficult in a single paper notepad. Even with a pencil, we might have to encounter a lot of erasing when it comes to managing notes. A digitized version of the notepad can actually make use of a filesystem where directories can represent Subject/Topics and their contents user's manuscripts. Apart from this a completely new feature of zooming becomes accessible. This feature removes the restriction of a page limit upto a certain point. Higher Data Storage is naturally an addon to the digital counter part of a paper notepad. Mobility can be guaranteed using customized batteries. Since the device, specific to the implementation, has a touch screen, so the look and feel of a paper copy is replicated closely and thus enables users to adapt to the newer digital environments, quickly. In terms of usage following advantages can be postulated: While the size of a paper page may restrict a user to write/draw abnormally from their original freehand(smaller or larger), using the device zooming in and out capabilities they can always stick to their comfortable selves.

Users may always reuse their notes by copy/paste"ing" a portion or whole of them without having to re-write everything. Additional features include that users can render their handwriting to projectors thus enabling large no. of people to see it.

4. IMPLEMENTATION METHODOLOGY

Now that the design and the device's significance has been cemented, its high time to lay down the road to implementation. As seen in Section 2, open source technologies are much more flexible so an appropriate development and deployment environment needs to be configured. Augmenting to it some power efficient techniques need to be utilized. Apart from deployment of the application, we also need to compare the results of our implementation design with the obvious ones and to establish software parameters for the application itself. Through virtualization, testing of environments and applications gets easy and hence its requirement. Finally all the little details of the elicited application product need to be pondered over.

4.1 The Open and Embedded Way : OpenEmbedded

For the ease of implementing the design on the device, one can use the method of cross-compiling source codes on a high-end host machine, to create binaries of embedded apps. A hunt for such a framework, revealed OpenEmbedded[2].

Openembedded offers an apt cross-compile environment which allows developers to create apps for embedded systems. With its portability, OpenEmbedded can cross-compile and generate builds for various machines(mini2440 particularly in our case). Openembedded has a pre-specified tree structure which has configurations for many open source applications and development environments. All one need to do in order to start building is learn two things:(a) How to use a tool named bitbake and (b) How to create recipes for OpenEmbedded.

Bitbake is a tool which, for given recipes, does particular jobs like compiling, building or removing applications. It serves as the basis of OpenEmbedded. Consider a scenario where we wish to generate a jffs (Journalling Flash File System) image of the desktop environment namely opie(Open Palmtop Integrated Environment which is quite a popular embedded desktop environment) for a device, say xyz, which is supported by OpenEmbedded. Now a non OpenEmbedded approach would be to download all sorts of dependencies sources for the opie as well as its own source and cross compile everything which precedes an image generation of the "built" material. One also needs to be careful about device specific parameters while cross compiling the sources. Now with OpenEmbedded support all one has to do is this:

Table 1. Unix Command for "bitbake"ing an app.

```
localhost@localhost$ bitbake opie-image
```

where all sorts of device specification is simply to be put in a configuration file. Next is just wait and watch scenario, since bitbake will resolve all the sources and their download urls, shall fetch them, and finally compile and deploy them. Therefore to use preexisting open softwares one doesn't have to be too careful.

To add one's own source to be used by OpenEmbedded recipes need to be created, which are nothing but ".bb" python files having a particular syntax so as to be resolved by bitbake. Thus for deploying the intended notepad application, the following mandatories had to be considered:

1. Our application itself is the one and only GUI on the device(for resource savvy issues).

2. A device specific Qt Embedded SDK for creating the desired notepad app.

3. A deployable image of a file system (preferably jffs or cramfs) which encapsulates merely the core requirements for running the notepad app.

4. The above stated image recompiled for virtualization purposes.

The first one at the list is to be hand-coded as will be described in 4.4 and the remaining three are simple "bitbake"s.

4.2 Power Efficiency

In all embedded devices, power plays a major role, rather it proves to be a constraint in developing all applications. The reasons for that are quite obvious, mostly because they are battery powered and the longer the same battery runs[4], the better for the device. After all no body likes to keep their mobile devices plugged in for charging. They ought to be "mobile". Limited space or weight to hold memory as well as the processors also asks for better management of memory and processing.

Now, since we zeroed on the device Mini2440, which works on the ARM processor, and has a fixed hardware design, we are limited to engage the implementation only to the software that suits our design. Catering our development to this path, we can go for what is so called "Green" Software Design. The software in the device consists, broadly, of the following three parts. (I) The Linux kernel, (ii) The Windowing system, and (iii) The Notepad app. So, We shall look into a few techniques available for possible optimization in all these parts.

4.2.1 The Linux Kernel

The major point about the proposed design is its simplicity. So the additional features of the linux kernel, which are specific for intensive applications, can be stripped off leaving those which are necessary. For instance, Kernel is termed tick-less, if its timer is reimplemented so that the scheduler doesn't wake up the processor again and again. It augments the power-saving capabilities for the systems with dedicated applications.

4.2.2 The Windowing System

The required dependencies for the windowing system QTE as well as the notepad application are compiled using GCC, there are a few techniques available for optimization so as to increase the battery life.

(i) Many optimizations are provided by the cross compiling toolchains, like CodeSourcery etc.

(ii) Compiler optimization achieved through -O switches can enhance the speed as well as memory requirements.

(iii) GCC also allows direct assembly language codes to be incorporated in between the C/C++ code, which can efficiently be used to utilize processor-specific instructions.

Further, [7] talks about a low power front-end which uses a block-aware instruction set, [8] discusses about an architecture-level power optimization. Similarly, QTE can also be tuned for optimum performance as described in [1] which include altering

- Programming Style
- Static vs. Dynamic Linking
- Alternative Memory Allocation
- Bypassing the Backing Store

4.2.3 The Notepad app

The programming language used for development of the notepad app is C++, so several measures can be taken to create power-aware codes. For instance, the source code is optimized as per the instructions given in [3] ,

- Modulo 2 division to be implemented on unsigned int; owing to the fact that div16s requires the extra sign correction as compared to div16u in ARM compilers
- Modulo operation is avoided, instead the conditional statements are utilized, wherever possible.
- When several conditions are required, use of the grouped conditions is preferred over separate if statements, as it utilizes the ARM way of handling with conditions.
- All ARM instructions can be conditionalized. So to check the range of a number x between xmin and xmax, the code is written as

```
(x - xmin) < xmax
```

- A local copy of global variables is made, so as to allow the compiler to use registers for them. For the same reason, the variables passed as references are copied to a local variable, and later the change is reflected back in the end.
- Use of int in place of short or char, as the compiler by default uses 32 bits in each assignment.
- Function definitions to be put above their call.
- No of arguments to be limited to 4, so that the compiler can use registers in stead of placing them on to the stack.
- In use of pointer chains, the references are reduced to single level so as to avoid multiple memory lookups. For example,

```
z1 = a → b → x;
z2 = a → b → y;
```

is replaced by

```
temp = a → b;
z1 = temp → x;
z2 = temp → y;
```

4.3 Virtualizing things up

We need to test and compare the notepad app with other possible app builds. Now instead of serially testing each and every environment by burning it on the device it is much better to test them virtually. For this purpose we need a framework which can emulate a genre of devices on a high-end host machine. With a bit of searching it was revealed that specifically for ARM emulation there are 3 widely used emulators(opensource) around, namely → Softgun, Qemu and Skyeye. Among these Qemu[12] is the arguably most versatile and more rigorously maintained than its counterparts. Qemu supports emulation of many machines including popular ones like ARM, Atmel , Amiga etc. Our device mini2440 is ARM based and also falls under the supported category. With this it was decided to set up virtualization through Qemu. Setting up Qemu was nothing more than a :

Table 2. Unix Command for "bitbake"ing qemu.

```
localhost@localhost$ bitbake qemu
```

Now qemu needs the following parameters so as to start emulation:

- machine name specifying the device for which emulation needs to be done, if the device is supported than it's platform (ARM etc.) is automatically decided; in our case it is Mini2440 or versatile[ab/pb] (for a work around if Mini2440 fails)
- the file system image for the machine; we use ext2 (generated by bitbake as mentioned in 4.1) in our case since jffs2 is not supported on Qemu.
- kernel image generated for the machine (zImage or uImage, for instance), it is usually generated along with the above stated image when "bitbake"ing.
- specifying what sorts of external devices need to be emulated along with the machine like keyboard, touchscreen, mouse etc; In our case, we require only the touch screen as the only I/O device.

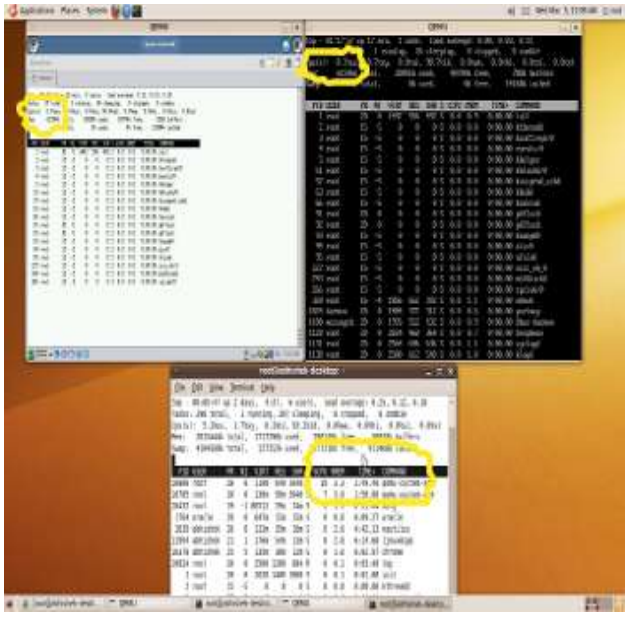

Illustration 4: Virtualizing Hardware

There are many other options supported by qemu like tftp-boot and telnet monitoring but we shall stick to the fundamental usage. For the testing purposes, we can generate images (as described in section 4.1) and emulate them so as to compare

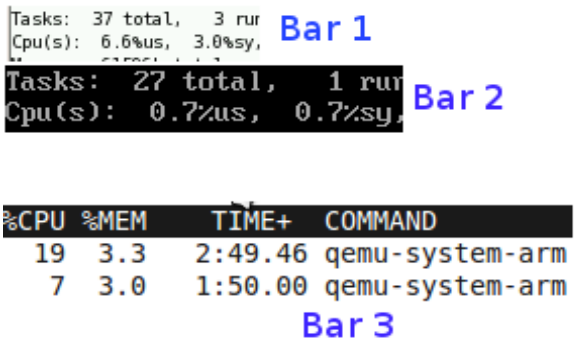

Illustration 5: Performance Results

the performance of the notepad app on desktop and non-desktop environments respectively.

4.4 The Final Application

Qt provides a bunch of features that we can use for the development of the simple notepad application. The following classes are used for the same:

- QPluginLoader : A plugin is a dynamic library that can be loaded at run-time to extend an application.
- QDirectPainterDirect: which enables access to the underlying hardware in Qt for Embedded Linux
- QWSWindow which encapsulates a top-level window in Qt for Embedded Linux
- QRasterPaintEngine, which enables hardware acceleration of painting operations in Qt for Embedded Linux
- QSvgGenerator which provides a paint device that is used to create SVG drawings and QSvgRenderer, which is used to draw the contents of SVG files onto paint devices

5. EXPERIMENTAL RESULTS

Here we describe some of the stats relating to virtualization of the device. In the image below there are two instances of the virtual environment of a completed embedded system running. The one on the L.H.S has a fully-blown desktop and GUI. Another on the right has been stripped of its GUI. And we display the results noted down both on the host as well as the target systems below.

The magnified yellow areas are shown above. The following data accounts for the scenario:The first bar(magnified from the LHS of the desktop screenshot) displays the CPU consumption of the target device when it is encompassed by a desktop(6.6% CPU usage). As in the case of the second virtual instance of the target device without the GUI, the CPU consumption drops down to 0.7%. Therefore it's quite safe to state that a desktop will definitely increase the processing requirements of the notepad environment and similar the case with power consumption. Therefore a complete stripped down version of the notepad app should be developed.

The third bar displays the memory and processing requirements of the virtual environments on the host. The one with the GUI is consuming about 19% of the CPU and is taking up .3% more memory than the non GUI instance. The host system has 2048 Mb of RAM and .3% accounts to 6 Mb usage. On an embedded system these many Mbs can be utilized for other creative purposes and hence the requirement of no GUI. Apart from the design requirements, application specific advantage is also experimented. We basically speak in terms of storage restrictions and hence we now compare SVG with raster graphics.

Illustration 6: Vector 'a'

Illustration 7: Raster 'a'

PNG Snapshot of letter 'a', magnified to same level from original(800%), in SVG and raster graphics respectively

The letter 'a' in the above snapshots clears out the advantage of using SVG as the fundamental image format for the app. Moreover the image shown below is formated in SVG and raster graphics.
A sample Black & White image requires 837 bytes when stored in SVG format with compression(SVGZ) and 13.1KB with raster graphics rendering. Thus SVG can cater its utilization effectively in the perspective of both the user and the system.

6. CONCLUSION & THE WAY AHEAD

This paper presented an idea to develop an eco-friendly device. having a simple but rarely implemented design in its core. We discussed about a candidate design that uses nothing novel in terms of technology, yet yielding a potential product. It has also been illustrated how different measures can be taken to optimize the functionality of the device.

As the notepad app gets developed it paves a way for encouraging scaling down of hardware. In the end we need a device which is both efficient and "cheap". The cost more or less depends on the hardware and hence the need for scaling down. With advancement in hardware the original designs may be reused and solution can be even better and yet cheaper.

7. ACKNOWLEDGMENTS

Our thanks to ACM SIGCHI for allowing us to modify templates they had developed, to our college for providing us the platform to research, and to OpenEmbedded developers who guided us through the alleys.

8. Glossary

PDA – Personal Digital Assistant
netbook – small, lightweight and inexpensive laptop computers suited for general computing and accessing web applications.
FriendlyARM – A category of devices which have an ARM processor and are good replacements of PDAs.
WinCE – embedded windows OS.
QT – GUI framework for linux based systems.
Qtopia – OS developed by QT developers with QT as GUI.
app – shorthand for application.
MIDlet – java apps for mobile based systems.
Distro – family of OS that uses linux as kernel.
param – shorthand for parameters.
QTE – QT framework for embedded systems.
Virtualizing – using the virtual replacement of an embedded device.

9. REFERENCES & CITATIONS

[1] Qt for Embedded Linux Performance Tuning, [as in March 2010] http://doc.trolltech.com/4.4/qt-embedded-performance.html [as in Feb 2010]

[2] OpenEmbedded [as in Feb 2010] http://wiki.OpenEmbedded.net/index.php/Main_Page

[3] Tajana Simunic, Luca Benini, Giovanni De Micheli, 1999. Energy-efficient design of battery-powered embedded systems. ACM Press, New York, NY, 212 – 217. DOI = http://doi.acm.org/10.1145/313817.313928

[4] Daler Rakhmatov, Sarma Vrudhula, Deborah A. Wallach 2002. Battery lifetime prediction for energy-aware computing. *In Proceedings of the international symposium on Low power electronics and design (Monterey, California, USA, 2002). ACM Press, New York, NY, 154-159. DOI=* http://doi.acm.org/10.1145/566408.566449

[5] *Qt vs Java* http://turing.iimas.unam.mx/~elena/PDI-Lic/qt-vs-java-whitepaper.pdf

[6] FriendlyARM http://www.friendlyarm.net

[7] Brown, L. D., Hua, H., and Gao, C. 2007. A low power front-end for embedded processors using a block-aware instruction set. In Proceedings of the 2007 international conference on Compilers, architecture, and synthesis for embedded systems (Salzburg, Austria, 2007). ACM Press, New York, NY, 267 – 276. DOI= http://doi.acm.org/10.1145/1289881.1289926

[8] David Brooks , Vivek Tiwari , Margaret Martonosi, Wattch: a framework for architectural-level power analysis and optimizations, Proceedings of the 27th annual international symposium on Computer architecture, p.83-94, June 2000, Vancouver, British Columbia, Canada [DOI = http://doi.acm.org/10.1145/339647.339657]

[9] Daily Green http://www.thedailygreen.com/environmental-news/latest/7447 [as in March 2010]

[10] Newspaper Recycling http://www.uaacog.com/Newspaper_Recycling.htm [as in Feb 2010]

[11] IES, NCES http://nces.ed.gov/pubs2001/overview/ [as in March 2010]

[12] QEMU http://www.qemu.org [as in Feb 2010]

[13] Scalable Vector Graphics http://www.w3.org/Graphics/SVG/ [as in Feb 2010]