A Light-Weight Text-Based User Interface Module for Small Embedded Systems

Jonghyuk Park and Nakhoon Baek*

Kyungpook National University, Daegu 702-701, Korea
clsveris@gmail.com, oceancru@gmail.com

Abstract

In these days, smaller devices are possible to show graphics output, through attaching customized LCD panel displays. Our target system is a small embedded system with an LCD panel. As a typical low-tier embedded system, it has a low-end CPU and restricted memories. We aimed to build-up a full-scale 2D graphics module on it. After analyzing system requirements, we found that the most hardiest restriction is the size of available memory. As an alternative to the full-scale modules, our design shows a TUI (text user interface) model, which is similar to the Unix Curses library and/or primitive graphics system used on earlier PC user interfaces. Our design has a drawback of restricted controllability of 20×20 character resolutions on the screen, rather than 240x320 pixel resolutions, while it enables pop-up menus and screen back buffering, with less than 7K extra memory usage. User Interface features including pop-up menus, push buttons, and customizable windows are also provided.

Keywords: *Graphics system; embedded system; user-interface features*

1. Introduction

Recently, we have more and more need for graphical interfaces for even low-tier embedded systems. Typically, these low-tier embedded systems have a small amount of memory. As an example, our target system has 48K SRAM and 16KB ROM, with 256KB NAND flash memory. This amount is actually sufficient for their usual application programs and operating software.

As an inexpensive graphics output solution, dummy LCD panels are often used. As an example, we used a QVGA 240×320 portrait LCD panel with (6,6,6)-color support. This panel has a single graphics buffer of 168KB GRAM for 18bit/pixel, only for a single screen. To efficiently support user-interface features on this kind of low-tier systems, we present our implementation of light-weight text-based user interface module.

More precisely, we represent an overall design of a specialized small-footprint user-interface module for these kinds of low-tier small-size embedded systems. Our example target system, as shown in Figure 1, has the following features[1]:

- ARM Cortex M3 Processor
- 256KB NAND Flash memory

* corresponding author: Nakhoon Baek, oceancru@gmail.com

This research is supported by Ministry of Culture, Sports and Tourism(MCST) and Korea Creative Content Agency(KOCCA) in the Culture Technology(CT) Research & Development Program(Immersive Game Contents CT Co-Research Center).

This research was also supported by Kyungpook National University Research Fund, 2012.

(a) an embedded board with an LCD display (b) the LCD display with 240×320 pixels

Figure 1. Our Target System

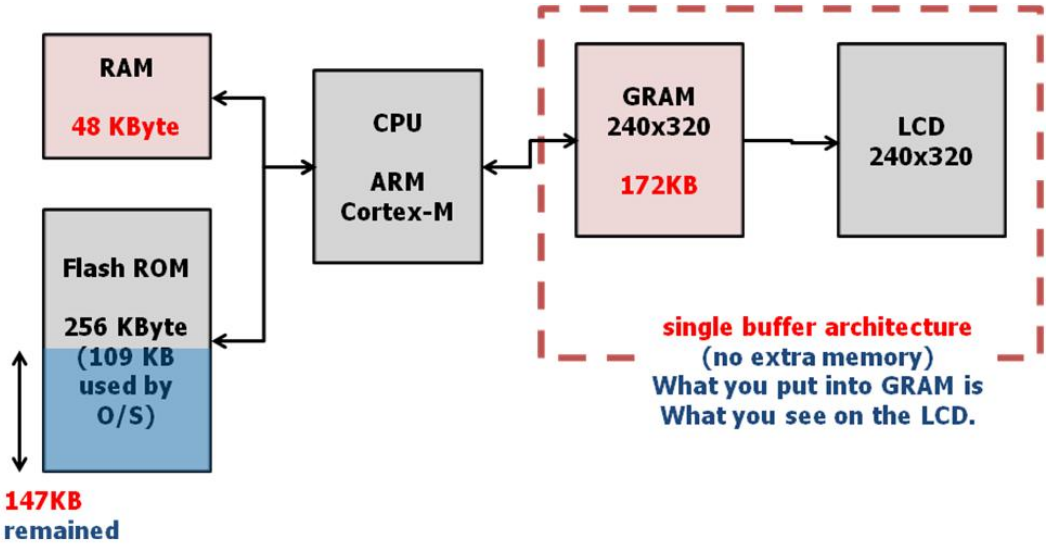


Figure 2. Overall Memory Layout

- 48KB SRAM
- 16KB ROM (for boot-loader)

For its graphics output, an LCD panel display is integrated with the following features:

- 2.8 inch TFT color LCD
- QVGA resolution: 240 × 320
- 172KB GRAM to support 18 bits / pixel

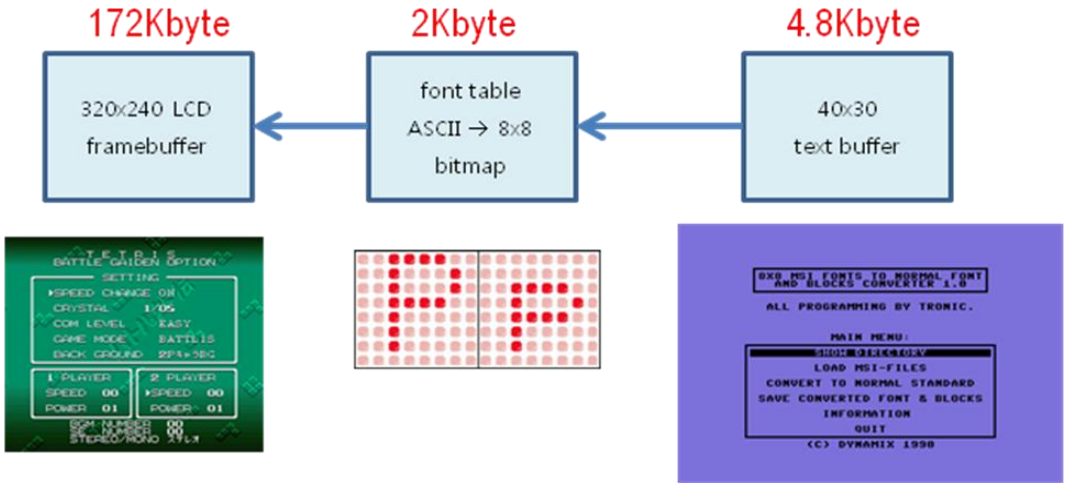


Figure 3. Our Text-based Interface Module Design

Requirements on this system was to build up a middleware-level graphics system on it. Although its device-driver level graphics primitives are available, they only provide direct access to the GRAM (graphics RAM) on the LCD panel and extra simple graphics routines including line and circle drawing, bitmap display, etc. Our analysis and design steps are followed.

2. Analysis and Design

On the target system, a real-time operating system, named Ubinos [2] is used. We carefully checked the footprints of all the files, and finally found that totally 109KByte Flash memory is required by the fundamental Ubinos system files and low-level device drivers including the LCD display drivers. Thus, less than 147KBytes are available for other middleware and user application programs. This small size of available memory, we need to design our graphics system as small as possible.

At the very beginning, we planned to implement a full-scale 2D or 3D graphics system, such as small-X [3] or Qt [4]. In contrast, The most important restriction on this system was the size of frame-buffer and the main memory.

With the standard QVGA resolution, it needs 240×320 pixels for a single screen. Since the LCD display only supports 6 bits for each of RGB components, it requires 18bits per pixel. As the typical small LCD systems, we can substitute this 18bit color system with 16bit system, with (5,6,5) components for each of RGB color values.

Even with (5,6,5)-color system, we need totally 240 × 320 × 2bytes = 153,600 bytes, or approximately 153KBytes. In contrast, our target system only supports 48KBytes of RAM. Conclusively, we cannot save the whole screen at all. With this bare system design, we cannot support any pop-up menu displays, which require to save the underlying screen area to the memory temporarily.

As shown in Figure 2, the operating system and other user applications would seriously compete for this RAM area. We need to minimize the footprint of our system, especially less than 10KBytes, if possible.

After analyzing the application requirements, we found that a kind of character-based user interface may work on this system. Figure 3 represents our overall design for the text user

interface (TUI) system. We first adopted a ASCII font system. As an example, with 8×8 resolution fonts, it requires a font table of 256 entries for each 8 byte compressed character image. Thus, totally 2,048byte is used for the font table. Using 8×8 character images, the 240×320 LCD screen is now considered as 30×40 characters text buffer. It requires only 1,200 byte for the whole screen.

To support pop-up menus and other features, we need double buffering for the screen. Thus, extra 1,200 byte is used for this purpose. Now, totally $2,048 + 2 \times 1,200 = 4,448$ bytes are required to support full screen TUI with pop-up menu and screen save features.

This kind of TUI-based design shows the absolute strong point of small footprints. In contrast, it shows serious drawbacks: it cannot support the more detailed drawing facilities including line segments, circles, and bitmaps [5].

Our TUI system plans to use the standard curses API specifications [6]. Current curses library specification contains various features including sub-windowing, forms-style input, etc. At the first stage of our implementation, we plan to support the following API features:

- **initialize the TUI module**; initializes the whole screen for CUI-based output.
- **finalize the TUI module**; resets the terminal.
- **clear the screen**; clears the screen and place the cursor in upper-left corner.
- **move the cursor**; moves the cursor to the indicated row and column.
- **print a character**; writes the given character at the current cursor position, and move the next position.
- **insert a character**; insert a character; all characters to the right move one space to the right.
- **refresh**; updates the screen to reflect all changes.
- **delete character**; delete character at the current cursor position.

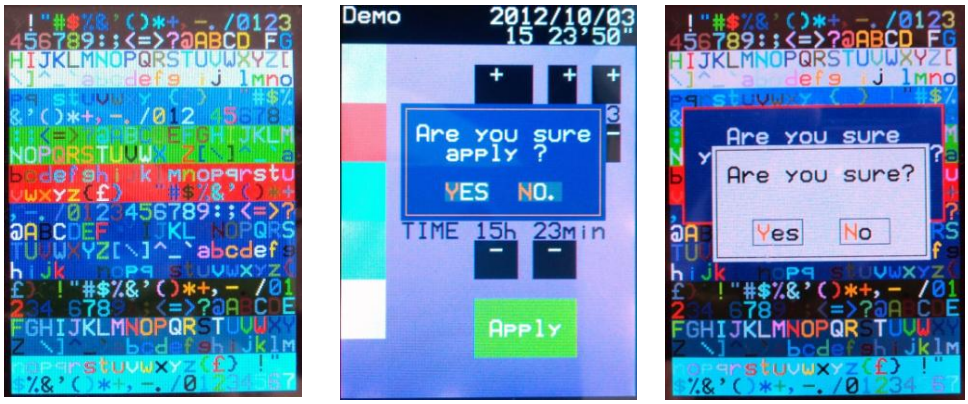
3. Implementation

To support pull-down menus and/or pop-up windows, double buffers are intuitively required. However, most low-tier embedded systems have strictly small memory size. For example, a naive back buffer implementation may need another 150K Byte RAM for the back buffer itself. Our design policy focuses on the minimization of the memory requirement, while keep the graphics facility.

First, we naturally introduce a text-based output system. Using 10×14 pixels fixed fonts, we arranged the output characters for every 12×16 pixels cell. Thus, QVGA 240×320 portrait LCD panel contains a 20×20 text layout, as shown in Figure 4 (a). Using this indirect text buffer, we can save the whole text-output screen into 400 byte memory, with extra $10 \times 14 \times 95 / 8 = 1663$ byte font ROM. It was a dramatic decrease from naive 150K Byte requirement.

Second, to express a variety of colors for the text characters and their backgrounds, we use a color palette with 64 colors. Figure 4 (a) shows various combinations of those foreground and background colors. To save the color information, we need extra 400 bytes for the indirect text buffer.

Using this foreground and background features, we can generate typical pop-up windows, as shown in Figure 4 (b). Multiple pop-up windows are naturally supported, as shown in Figure 4 (c). We also introduced extra decoration lines around the pop-up window, as shown



(a) our 20×20 characters layout with the 64-color palette

(b) a pop-up window displayed with boundary decorations

(c) multiple pop-up windows

(d) a calendar application

(e) a text-display application

(f) a qwerty-style keyboard interface

Figure 4. Selected Screen Shots from our Demonstration Programs

in Figure 4 (b) and 4 (c). This decoration lines are treated as the window properties. A set of sample applications are also developed as shown in Figure 4 (d), 4 (e) and 4 (f).

4. Conclusion

For our low-tier embedded systems with low-power LCD panel display, we found that any full-scale graphics processing systems are not suitable, mainly due to its very restricted memory size. After analyzing the system and also the user requirements, we finally designed a text-buffer based graphics processing system. In this system, we used a traditional text buffer. This design enables its full screen back-ups and pop-up menus with only very small extra memory areas.

In this paper, we present our implementation of a low-tier light-weight text-based user interface module. A set of example applications are also demonstrated. This system can be used for text-based pull-down menus and pop-up windows. Our next step is to integrate this simple text-based menu system with touch screen facilities.

Acknowledgements

This research is supported by Ministry of Culture, Sports and Tourism(MCST) and Korea Creative Content Agency(KOCCA) in the Culture Technology(CT) Research & Development Program(Immersive Game Contents CT Co-Research Center).

This research was also supported by Kyungpook National University Research Fund, 2012.

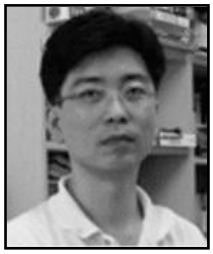
References

[1] Atmel, "SAM3S-EK Development Board User Guide", Atmel **(2011)**.
[2] Ubinos, http://www.ubinos.org/.
[3] Small Linux, http://www.superant.com/smalllinux/tinyX01.html.
[4] Nokia, "Qt: A cross platform application and UI framework", http://qt.nokia.com/.
[5] Wikipedia, "Color Graphics Adaptor", http://en.wikipedia.org/wiki/Color_Graphics_Adapter.
[6] N. Matloff, "Introduction to the Unix Curses Library", Dept. of Computer Science, UC Davis, **(2011)** April.

Authors

Jonghyuk Park

Jonghyuk Park is now a master student in School of Computer Science and Engineering, Kyungpook National University. He received his B.A. in Computer Science from Kyungpook National University. His interests include computer graphics and real-time embedded systems.



Nakhoon Baek

Nakhoon Baek is currently an associate professor in the School of Computer Science and Engineering at Kyungpook National University, Korea. He received his B.A., M.S., and Ph.D. degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST) in 1990, 1992, and 1997, respectively. His research interests include graphics standards, graphics algorithms and real-time rendering. He is now also the Chief Engineer of Mobile Graphics Inc., Korea.