A Compressed Frame Buffer to Reduce Display Power Consumption in Mobile Systems

Hojun Shim
School of CSE
Seoul National University
Korea 151-010

Naehyuck Chang[§]
School of CSE
Seoul National University
Korea 151-010

Massoud Pedram
Department of EE
University of Southern California
USA CA 90089

Abstract— Despite the limited power available in a battery-operated hand-held device, a display system must still have an enough resolution and sufficient color depth to deliver the necessary information. We introduce some methodologies for frame buffer compression that efficiently reduce the power consumption of display systems and thus distinctly extend battery life for hand-held applications. Our algorithm is based on a run-length encoding for on-the-fly compression, with a negligible burden in resources and time. We present an adaptive and incremental re-compression technique to maintain efficiency under frequent partial frame buffer updates. We save about 30% to 90% frame buffer activity on average for various hand-held applications. We have implemented an LCD controller with frame buffer compression occupying 1,026 slices and 960 flip-flops in a Xilinx Sprantan-II FPGA, which has an equivalent gate count of 65,000 gates. It consumes 30mW more power and 10% additional silicon space than an LCD controller without frame buffer compression, but reduces the power consumption of the frame buffer memory by 400mW.

I. Introduction

Power consumption has been a constant issue in computer system design at a range of levels from device to application. Many power reduction techniques have been developed for microprocessors and memory devices. High-level power reduction generally utilizes slack time to reduce the power consumption. Thanks to previous research, a designer has been able to satisfy the power requirements of a microprocessor and memory system when their utilization is not high, as is often the case with hand-held computers used in interactive applications. Now we are faced with a different situation since the energy requirement of modern color TFT LCDs is high. High-resolution, high-color LCDs are also involved with their own large-capacity frame buffer memory and thus large power consumption. Worst of all, there is no explicit slack time in display systems, regardless of the computational burden, as long as the display is turned on. Display systems are among the most power-hungry components in embedded systems.

System-level techniques for power reduction in display systems have been recently introduced: variable duty-ratio refresh, dynamic color-depth control, and brightness and contrast shift with backlight luminance dimming [1]. Among these, dynamic color-depth control saves power consumption in the frame buffer and associated buses as long as we are prepared to accept noticeable color fidelity degradation.

We introduce a virtually lossless frame buffer compression scheme, allowing powers to be saved without display quality degradation. Our scheme reduces the activity of the frame buffer and associated buses during sweep operations, and thus reduces the power consumption of the frame buffer for most hand-held applications. For energy evaluation, we use a high-accuracy approach based on precise energy measurement, characterization and trace-driven analysis. Starting from a basic compression scheme, we introduce an adaptive, incremental re-compression scheme to maximize energy gain under frequent partial content changes of the frame buffer. We show the energy reduction achieved by the proposed schemes for various hand-held applications, using a prototype equipped with an FPGA implementation of the LCD controller.

[§]Correspondence should be addressed to:

Prof. Naehyuck Chang
School of Computer Science and Engineering
Seoul National University
Shilim-dong, Kwanak-ku Seoul, Korea 151-010.
Phone: +82 2 880 1834, Fax: +82 2 886 7589
Email:naehyuck@snu.ac.kr

II. Power consumption of a frame buffer

Fig. 1 shows where the power goes in a typical hand-held computer with a color TFT LCD display. The power consumption of the LCD display system is significant. Our reference system is a typical palm-size PC or PDA, which is equipped with a 32-bit RISC CPU: the StrongARM running at 206MHz [3]. The main memory of this system consists of four K4S280832B-TC1L chips [4] from Samsung, with a bus length of $2''$ and an equivalent capacitance of 2.7pF. The memory data bus is buffered by the 74LVT245 transceivers from Fairchild with an I/O capacitance of 4.0pF; the bus-hold circuit in the buffer has an additional 0.5pF equivalent capacitance; the I/O capacitance of the K4S280832B-TC1L data ports is 5.3pF; the memory address bus is driven by the 74LVT244 buffers from Fairchild with an output capacitance of 4.0pF; the input capacitance of the address ports is 15.0pF, because four K4S280832B-TC1L chips are connected together. We perform cycle-accurate energy simulation with state-machine-based energy models borrowed from Shim et al [5]. The reference luminance of the backlight lamp is 30cd/m^2. We use a Toshiba LTM04C380K LCD panel as our energy model. This is a 4-inch 640×480 high-color TFT LCD display with a CCFL backlight [6]. We use the power consumption figure for the backlight system presented by Choi et al [1]. The power consumption of the LCD controller is discussed in Section 4.

When a system is waiting for a human reaction, the CPU is mainly sleeping and the main memory is primarily in the power-down state. However, the frame buffer memory and buses are busy at all times due to continuous sweep operations while the LCD panel is turned on. We do not include high-bandwidth RF communication adapters. Their power consumption depends not only on the application dependent but also on the user data, and thus difficult to quantify in general. Roughly speaking, during transmission, their power consumption will be similar to that of the total display system. Movie players are an extreme example of power-hungry battery-operated systems. Even in movie players, the display system dominates power consumption. A memory-intensive application, such as a warehouse manager consumes more power than an eBook or a map viewer, but the portion is still minor.

Emergent organic display systems illuminate themselves, thus avoiding a high-power backlight system [7]. In such a system, we may expect the power requirements of the frame buffer memory and associated buses to become more dominant (see Fig. 1).

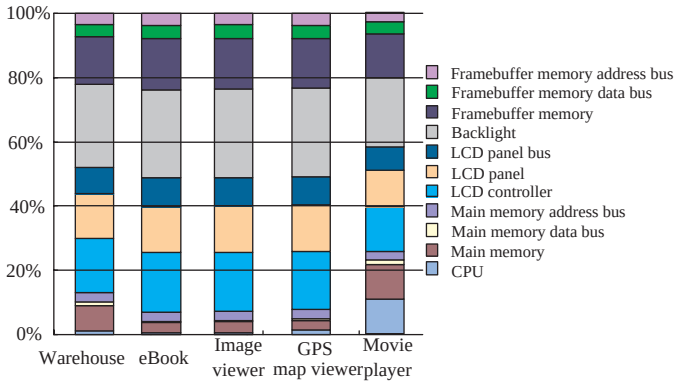


Fig. 1. The system-wide energy consumption of various applications with an FPGA LCD controller.

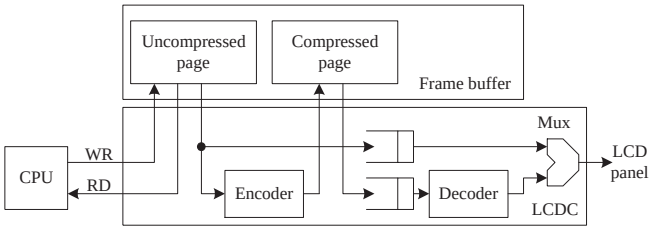


Fig. 2. The compressed frame buffer.

III. The Compressed Frame Buffer

A. Frame buffer compression

Many years ago, frame buffers were composed of conventional DRAMs. Dual-port video DRAMs used to be popular, as they fulfilled bandwidth requirements [8]. Today, SDRAMs or DDR (double data-rate) SDRAMs are typical. A cost-effective architecture shares the main memory with the frame buffer. The ancient Apple II computer used this structure to save memory cost. Modern StrongARM-based microcontrollers often use the same structure, but populated by high-bandwidth SDRAM memory devices. This revival of an old-fashioned architecture is limited to low-performance LCD display systems. Using a unified main and frame buffer memory, bandwidth may be unsatisfactory, but it can be enhanced by a compressed frame buffer. The *CompressDRAM* [9] is an integrated DRAM with compression and decompression hardware for graphics data. It uses simple run-length encoding (RLE) or differential pulse code modulation (DPCM) to enhance bandwidth. The similar technique using segmented encoding of graphic data was also introduced by Huang et al [2]. Segmented encoding technique uses extra hardware logic to run a run-length encoder, a run-index encoder, and a bit-map encoder to compress graphic data. But both of those designs were solely focused on performance enhancement without considering of power consumption.

Our compressed frame buffer is aimed not only at reducing the number of frame buffer accesses, but also power consumption. We use simple RLE and its variations. Fig. 2 illustrates the basic structure of our compressed frame buffer. We use a separate frame buffer memory of an almost standard structure (except some low-cost products). It has compressed and decompressed pages. The frame buffer is transparent to the common device drivers and graphics managers and thus maintains compatibility because the microprocessor can access the decompressed pages only. The compressed pages are managed by the LCD controller, which compresses the contents of the decompressed pages during the sweep operation. The only additional frame buffer accesses for compression are write-back operations on the compressed pages.

The energy consumption of the frame buffer and associated buses is proportional to the number of frame buffer accesses during the sweep operation (as long as a microprocessor does not access the frame buffer too frequently). The number of frame buffer accesses is determined by the screen resolution, the sweep rate and the color depth. As these are generally constant, the number of frame buffer accesses is naturally constant in a conventional architecture. Frame buffer compression reduces the number of frame buffer accesses and thus saves power.

The compression algorithm must be simple and not consume much resources to avoid counterproductive energy overhead. The encode and decode processes must be managed by simple hardware embedded in the LCD controller, so as not to involve the microprocessor or the memory system. We use a simple, single-pass compression algorithm based on RLE, whose data structure is illustrated in Fig. 3. The encoder compares the new pixel data with the previous one and increments a length counter if they are equal. We name it RLE16 by extension of the well-known RLE8 algorithm, which uses an 8-bit chunk.

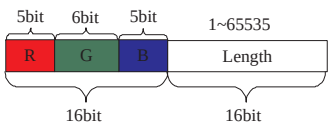


Fig. 3. RLE16 compression.

B. Target applications

Frame buffer compression is more effective when the frame buffer is less frequently updated by the microprocessor. For fair comparison, we consider various hand-held applications. There is virtually no restriction on running desktop-like applications on a modern PDA or palm-size PC. We can find many applications at `http://www.microsoft.com/mobile/pocketpc` or at `http://www.pocketpcmag.com`.

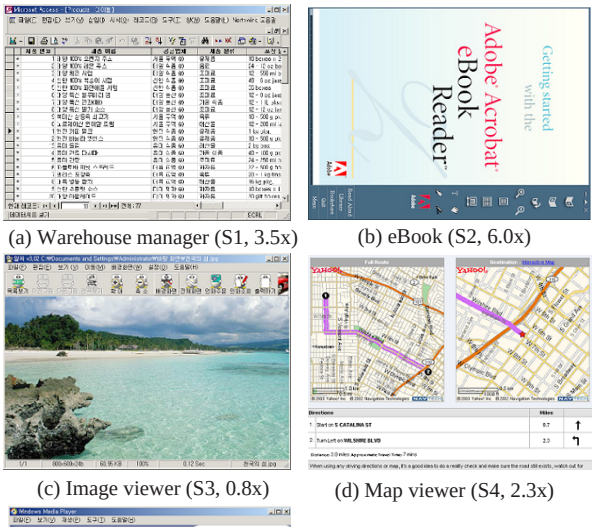
We can categorize the screen update models of applications into two classes: interactive and streaming. Applications with interactive screen update models change the display at the request of the user. Since human interaction is very slow when compared with the performance of digital systems, the sweep operation frequently refreshes the same data to the LCD panel. On the other hand, applications with streaming screen update models change the display much more frequently .

Fig. 4 shows typical screen shots of the target applications, denoted by S1 to S5 for brevity. Except for S3, the four screen shots show good compression ratio being achieved with only the simple RLE16 algorithm. But the large photo being displayed in S3 results in negative compression. We will give a detailed power analysis results in later sections. As a rule of thumb, power reduction is proportional to compression ratio.

We model the screen updates as independent Poisson processes. Each process has a different arrival rate and a different block size for updates. The location of updates is random, but in any case that has no impact on the energy consumption. Table I summarizes the Poisson processes. To make our evaluation more strict, we have assumed heavily loaded applications. But human interactions are still slow in comparison with the screen refresh rate, 60Hz, and thus the optimal configuration is robust with such variations in the model.

C. Adaptive RLE16

Since the situation shown in S3 (displaying the large photo) is common in target applications, we need to avoid negative compression with this sort of data though we do not expect



(a) Warehouse manager (S1, 3.5x) (b) eBook (S2, 6.0x)

(c) Image viewer (S3, 0.8x) (d) Map viewer (S4, 2.3x)

(e) Movie player (S5, 4.2x)

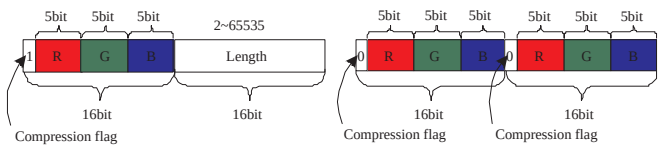
Fig. 4. Screen shots of applications and the associated compression ratios using the simple RLE16 algorithm.

TABLE I
SCREEN UPDATE MODELS.

Application	Update block size (pixel)		Arrival rate (λ: 1/sec)
	Horizontal	Vertical	
Warehouse manager	640	480	1/5
	160	120	1/3
	8	16	2
	8	16	5
eBook	640	480	1/10
Image viewer	640	480	1/5
Map viewer	300	300	1/3
	300	300	1/3
Movie player	320	240	20

TABLE II
COMPRESSION RATIO OF THE SIMPLE RLE16 AND THE ADAPTIVE RLE16.

Screen	S1	S2	S3	S4	S5
Simple RLE16	3.50	6.04	0.84	2.30	4.15
Adaptive RLE16	4.93	8.95	1.46	3.14	7.11



(a) When the length is more than two (b) When the length is one

Fig. 5. Adaptive RLE16 compression.

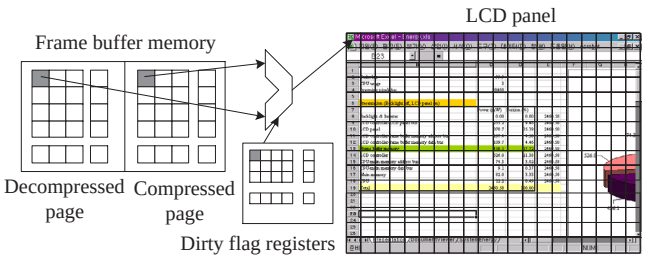


Fig. 6. Incremental re-compression for partial frame buffer update.

a good compression ratio. Adaptive RLE16 selectively compresses the pixel data only when at least two consecutive pixels have the same color. This guarantees that the size of the compressed data does not exceed that of the original. For adaptive RLE16, we need a one-bit flag that represents whether the pixel data is compressed or not. Generally, 16-bit color allocates one more bit for green, but this does not have much meaning. We use the LSB of the green data as our flag, but the quality of the color remains almost the same. As shown in Table II, adaptive RLE16 always outperforms simple RLE16.

D. Incremental adaptive RLE16

So far, we have discussed the compression ratio of the screen images and introduced the adaptive RLE16 algorithm to avoid negative compression. This spatial behavior largely determines the compression ratio and thus energy reduction. However, the actual energy gain is determined by the energy gain due to compression minus the energy overhead incurred by compression.

The compression overhead is generally very small and this will be confirmed in Section V. However, the compression overhead may not be negligible if the screen is frequently updated. An eBook and an image viewer update the frame buffer only when a page change occurs; but a movie player frequently changes part of the screen.

As described so far, our algorithm must compress the whole screen even though only one pixel has been updated. Fig. 6 illustrates an architecture for incremental re-compression. Incremental adaptive RLE16 divides the screen into several zones and maintains dirty flags to specify newly updated zones. In a real implementation, additional compress flags are necessary for the LCD controller to determine whether each block is being compressed or not. Spatial complexity is determined by the number of blocks which also determines the number of flag registers. Both the block width and the block height affect the number of blocks. Moreover, determination of the width and the height is related to the energy reduction ratio; we will discuss this issue in Section V.

IV. IMPLEMENTATION

We implemented a conventional LCD controller with an SDRAM frame buffer memory and an XC2S-150FG456 Xilinx FPGA. The LCD controller includes a frame buffer controller, a local bus interface, a sweeper, a video timing generator, and a bus arbiter. We used Block RAM primitives to implement internal storage such as FIFO memory. Block RAM primitives are dedicated blocks of true dual-port RAM; avoids abuse of the logic resources to implement the internal memory. In total, 857 slices, 774 slice flip-flops 182 IOBs, 2 GCLKIOBs and 2 DLLs, which have an equivalent gate count of 62,771 gates, are used.

We added the frame buffer compression feature to the LCD controller. We implemented dirty flag registers, compress flag registers, an encoder and a decoder as shown in Fig. 7, using

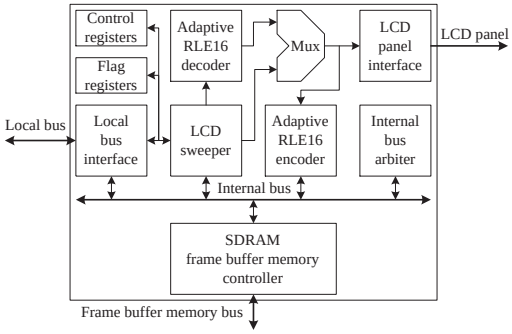


Fig. 7. Block diagram of the compressed frame buffer LCD controller.

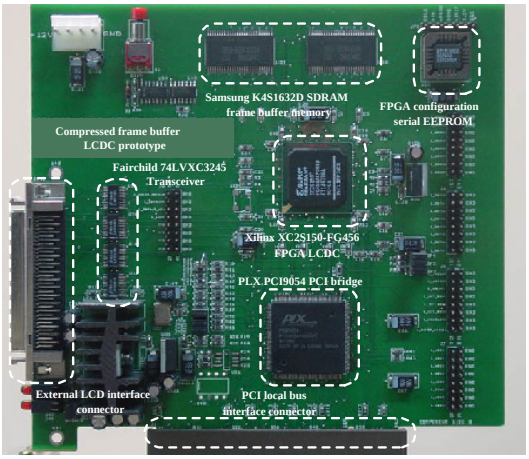


Fig. 8. Compressed frame buffer LCD controller prototype in the form of a PC card with a PCI local bus interface.

Block RAM primitives for the flag registers. Several pipelined dividers and multipliers were generated by the Xilinx Core Generator to implement the encoding and the decoding processes of incremental adaptive RLE16. We integrated control registers for enabling, disabling and dynamic configuration of the frame buffer compression allowing parameters such as block width and block height to be varied. Implemented in this way, the controller occupies 1,026 slices, 960 slice flip-flops, 182 IOBs, 2 GCLKIOBs and 2 DLLs, with an equivalent gate count of 65,323 gates. The area overhead is around 10%.

We implemented a prototype with the FPGA LCD controller, as shown in Fig. 8, and verified the compressed frame buffer and its power consumption. The LCD controller with the frame buffer compression feature consumes 30mW more power than a conventional design, which consumes 470mW on average. The prototype is a PCI local bus card connected to a Linux PC. We used a PLX PCI9054 PCI bridge to interface between the PCI local bus and the local bus of the LCD controller; and four Fairchild 74LVXC3245 transceivers to interface between the FPGA LCD controller and the LCD panel. We integrated a test platform with a LP064V1 [10] which is a 640×480 resolution 18-bit color TFT LCD from LG. Philips, as shown in Fig. 9. We ran various kinds of applications on the test environment, with a custom frame buffer device driver for the prototype, all running under the Linux kernel 2.4.17.

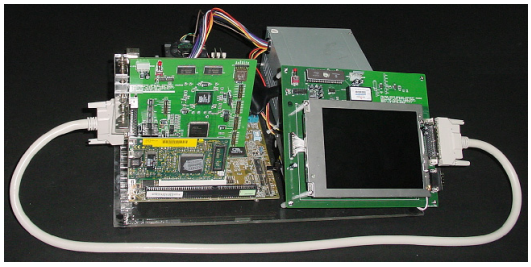


Fig. 9. The test environment for a compressed frame buffer LCD controller prototype.

V. Performance Analysis

A. Compression gain

Incremental re-compression requires a small block width. Block width directly affects the compression ratio as well as the implementation cost. Fig.10 shows energy saving versus block width.

As Table I denotes, the eBook and the image viewer require no partial screen updates, but the whole screen updates every 10 seconds or 5 seconds, respectively. Thus the energy gain is solely affected by the compression ratio, and that increases as the block width increases. On the other hand, the block height has no impact on the energy gain, although it increases the required number of the dirty flag registers.

Frequent partial screen updates are made by the warehouse manager. Frequent updates generally occur in small areas, but the warehouse manager frequently requires large large partial screen updates, for instance for drawing pop-up menus. Partial screen updates invalidate the whole contents of the blocks in the updated area and thus the sweeper must read the decompressed page, even though some of the contents of the block are not actually dirty. This results in low energy gain, even though the compression ratio of the image is high (*i.e.* larger block width does not enhance the energy gain although it enhances the compression ratio). Theoretically this may occur when the block height is greater than two. However, as illustrated in Fig. 10, block heights of less than 32 pixels do not noticeably degrade the energy gain. Only extreme cases, such as 240-pixel or 480-pixel block heights, degrade the energy gain, as the block width exceeds 128 pixels.

A movie player may update the screen up to 30 times a second. We analyze a movie player with 320×240 resolution running at 20 frames per second. We find that a block width of 128 pixels maximizes the energy gain. Although a block width greater than 128 pixels increases the compression ratio further, the energy gain does not increase any more since the proportion of dirty blocks increases significantly.

B. Compression overhead

Since compression is performed by the LCD controller during the sweep operation, the overhead is trivial. But we should take into consideration the energy overhead with respect to FPGA implementation, which is 30mW for frame buffer compression, as mentioned in the previous section. A measurable additional overhead in time and energy only occurs when the LCD controller updates the compressed frame buffer. Each update consists of one or two consecutive DRAM writes. The number of total updates depends only on the compression ratio when there is no frequent partial screen update: the larger the compression ratio, the smaller the overhead. The graphs in Fig.11 (b) and (c) show this inverse relationship, using data depicted in Fig. 10 (b) and (c). Thus larger block width always results in a smaller compression overhead for the eBook and the image viewer.

On the other hand, frequent partial screen updates cause the compression overhead to be influenced by the proportion of dirty blocks. A larger block height severely increases the compression overhead as the block width also becomes

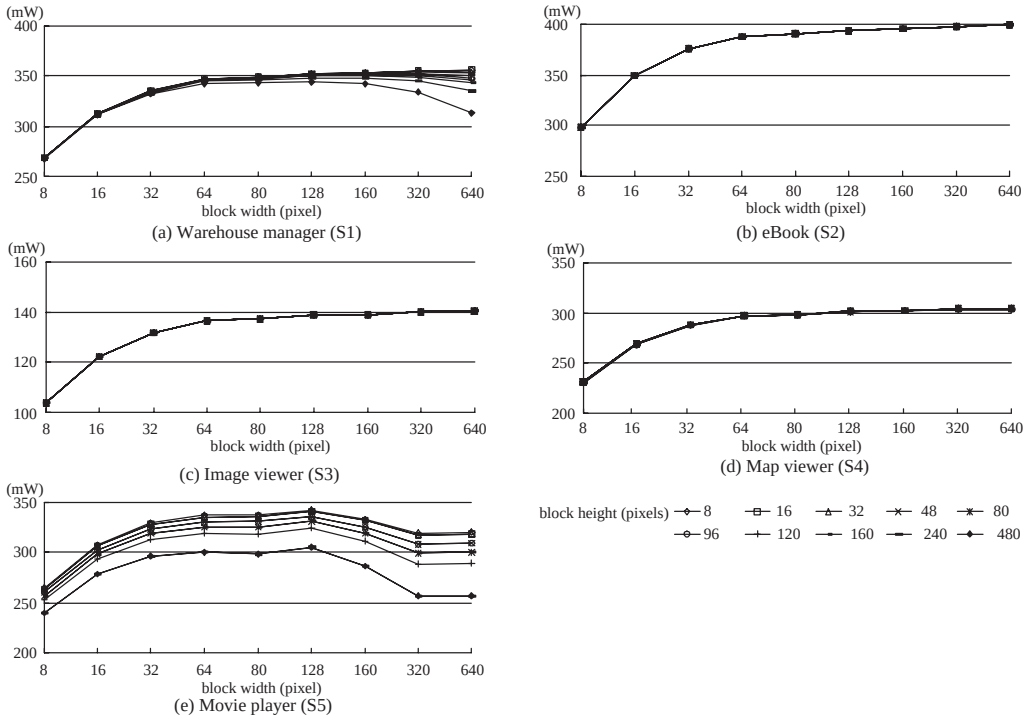


Fig. 10. Energy gain using the incremental adaptive RLE16 algorithm.

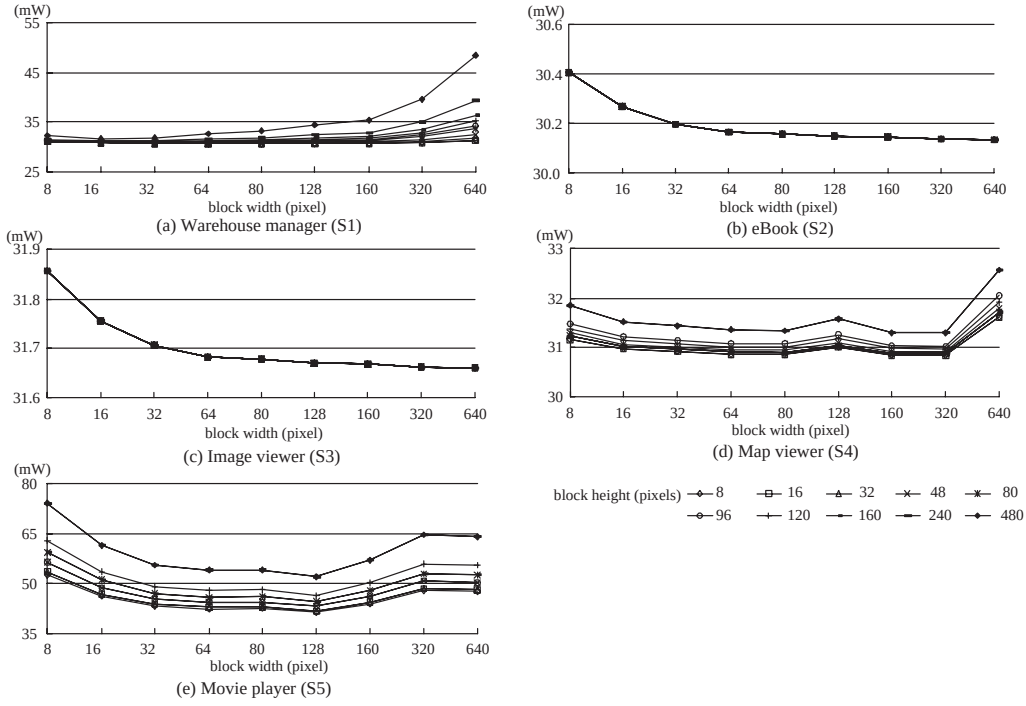


Fig. 11. Compression overhead using the incremental adaptive RLE16 algorithm.

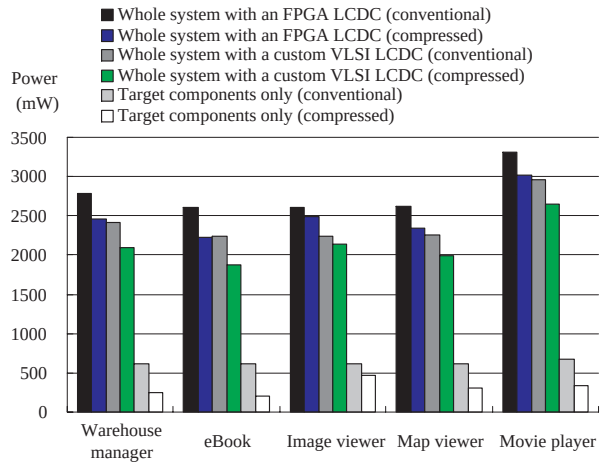


Fig. 12. Overall energy reduction achieved by the incremental adaptive RLE16 algorithm.

large. For example, a larger block height makes the LCD controller re-compresses several adjacent rows when the cursor is blinking on a row or new characters are being displayed on a row. A larger block width also increases the proportion of dirty blocks, although it does achieve a high compression ratio. Thus there is an optimal block width to minimize the compression overhead. The screen update models for the warehouse manager and the map viewer yield 64 and 80 pixels respectively as the optimal block width when the block height is eight. For the movie player, 128 pixels is the optimal block width to minimize the compression overhead. However, we must also take the energy saving into account together to select the optimal block width for the maximum benefit. In addition, Fig. 11 shows dividing the screen vertically into four is enough to reduce the compression overhead below the energy saved in the display.

C. Overall energy reduction

The overall energy gain is calculated by subtracting the energy overhead from the energy extended in compression. The compression ratio and the proportion of dirty blocks determine both the energy saving and the energy overhead. Each graph has its own optimal point that is determined by the block size associated with the screen update model. At this point, the compression gain is much larger than the compression overhead in all cases.

The block height affects the implementation cost because a smaller block height always reduces the compression overhead. At 640×480 resolution, an eight-pixel block height is feasible even with an FPGA. The optimal block size for the eBook and the image viewer (S2 and S3) is 640×8. The warehouse manager has the same optimal block size, despite the need for frequent partial updates. For the map viewer, 320×8 is optimal; this is because the refresh rate of the LCD panel is 60Hz, which is much faster than the screen update rate in normal applications. For these applications, the compressed frame buffer shows excellent performance without sensitivity to the block size. For the movie player, with extremely frequent partial updates, 128×8 is the optimal block size. Human interaction rarely exceeds 10 times a second. If screen updates are below 10Hz, a larger block width gives better performance. But when part of the screen is updated at close to 30Hz or higher, the optimal block width is determined by a trade-off between the compression ratio and the proportion of dirty blocks.

Fig. 12 shows the power reduction achieved by the compressed frame buffer. Compression reduces the power consumption of the target components, namely the frame buffer memory and its associated buses, by about 50% to 66%. In the extreme case, S3, there is about 23% reduction. When we implement the scheme with an FPGA as in this design, it saves about 10% to 15% of the total power consumption, except in the extreme case. This system-wide energy gain is smaller than real cases because an FPGA implementation of the LCD controller consumes much more power than would a custom VLSI implementation. Typical custom VLSI LCDCs [11] consume less than 1/3 the power of an FPGA implementation. With a custom VLSI implementation of the LCD controller, we could expect to save between 13% and 17% of the total system power consumption.

VI. Conclusions

We have introduced a compressed frame buffer LCD controller that dramatically reduces the frame buffer and associated bus activities for various target applications. We present an incremental adaptive re-compression algorithm based on run-length-encoding for on-the-fly, low-cost compression and decompression. We have intensively explored the design variables in terms of the block size, for the incremental re-compression. We derived the optimal configuration for several different screen update models. We have shown that the optimal block size is not sensitive to screen update rate.

Frame buffer compression reduces power consumption of the target components, the frame buffer memory and associated buses, by about 50% to 66%. It saves about 10% to 15% of total power consumption in the prototype, where the LCD controller is implemented with an FPGA.

References

[1] I. Choi, H. Shim, and N. Chang, "Low-power color TFT LCD display for hand-held embedded systems," in *Proceedings of International Symposium on Low Power Electronics and Design*, pp. 112 – 117, August 2002.

[2] H. Huang, J. Yao, and C. Chen, "Method and system for segment encoded graphic data compression," *United States Patent 5,748,904*, May 1998.

[3] *Intel StrongARM SA-1110 Microprocessor Developer's Manual*. Intel Corporation, http://www.intel.com, June 2000.

[4] *K4S280832 128Mbit SDRAM Datasheet*. Samsung Electronics Co. Ltd, http://www.samsungelectronics.com, August 1999.

[5] H. Shim, Y. Joo, Y. Choi, H. G. Lee, and N. Chang, "Low-energy off-chip SDRAM memory systems for embedded applications," *ACM Transactions on Embedded Computing Systems, Special Issue on Memory Systems*, vol. 2, pp. 98 – 130, April 2003.

[6] *LTM04C380K 4.0" Color TFT-LCD Module Product Information, http://www.tridentdisplays.co.uk*. TOSHIBA, Inc., March 2000.

[7] N. Kamijoh, T. Inoue, C. M. Olsen, M. T. Raghunath, and C. Narayanaswami, "Energy trade-offs in the IBM wristwatch computer," in *Proceedings of International Symposium on Wearable Computers*, pp. 133 – 140, October 2001.

[8] B. Prince, "A tribute to graphics DRAMs," in *Records of the IEEE International Workshop on Memory Technology, Design and Testing*, pp. 123 – 130, 1998.

[9] Y. Yabe, Y. Aimoto, M. Motomura, T. Takizawa, T. Miyamoto, T. Iwasaki, Y. Nakazawa, T. Fujii, M. Hamada, N. Nagai, and M. Yamashina, "Compression/decompression DRAM for unified memory systems: a 16 Mb, 200 MHz, 90% to 50% graphics-bandwidth reduction prototype," in *Digest of Technical Papers of IEEE International Solid-State Circuits Conference*, pp. 342 – 343, 1998.

[10] *LP064V1 6.4" VGA TFT LCD Preliminary Specification*. LG. Philips LCD Co. Ltd, http://www.lgphilips-lcd.com, Feb 1998.

[11] *SED1355 Technical Manual*. Epson Research and Development, Inc., http://www.epson.com, 1998.