UDC 004.92
DOI: 10.15827/2311-6749.20.3.2

Analysis of X Windows System and Hardware Interaction Methods

A.V. Roditelev [1], *Leading Programmer, rav@niisi.ras.ru*
K.A. Mamrosenko [1], *Ph.D. (Engineering), Head of the Center, mamrosenko_k@niisi.ras.ru*
A.M. Giatsintov [1], *Ph.D. (Engineering), Senior Researcher, giatsintov@niisi.ras.ru*
V.N. Reshetnikov [1], *Dr.Sc. (Physics and Mathematics), Professor, Chief Researcher, rvn@niisi.ras.ru*

[1] *Center of Visualization and Satellite Information Technologies SRISA RAS, Moscow, 117218, Russian Federation*

This article describes several methods that the X Windows System interacts with hardware. The basic 2D hardware acceleration operations were examined using the EXA architecture as an example. Xorg was tested with x11perf version 1.5 to examine the performance.

Testing was carried out in 5 modes, i.e. with 2D hardware acceleration and without. The test results lead to the conclusion that the use of 2D hardware acceleration with EXA is justified when high-resolution images are to be used or it is necessary to reduce the load on the CPU.

Performance testing with disabled hardware interaction (2D GPU + dummy EXA) showed that the use of EXA significantly affects the test rate as 2D GPU + dummy EXA driver performs better compared to the mode-setting driver only with a certain set of tests. It's worth noting that the use of EXA with the DRM stack has to be studied further, since, in most cases, scaling, pixmap copying, solid color filling, and other operations are processed by the CPU when using the DRM stack with the EXA_MIXED_PIXMAPS option. In certain situations, the absence of this option can lead to many visual artifacts in the form of unrendered areas of the image when updating the framebuffer.

Keywords: *EXA, Kernel mode setting, X.Org Server, embedded systems, Linux.*

As the volume of graphic information used in various fields of application of embedded systems is gradually growing, there appear more tasks with increased requirements for the performance of the GPU. An embedded system is broadly understood as a device that contains loosely coupled hardware and software components to perform one function, which is part of a larger system [1]. Systems on a chip are often used as the main platform for embedded systems. A system on a chip (SoC) is a functionally complete computing unit that includes system, peripheral and network controllers in addition to the CPU core [2].

SoC design involves software development, which solves the tasks of interaction between hardware and operating system (OS). Linux OS is one of the common operating systems used to control embedded systems. It is supported by a large number of processor architectures (ARM, MIPS, x86 and PowerPC), loadable modules. Being an open-source OS, Linux allows introducing the necessary changes into the base source code to support a specific SoC or device (i.e. to add protocols, drivers for new devices).

Displaying of information is characterized by specific features in many Linux-based embedded systems. Typically, the specific embedded system requires the development of a special driver for the display output controller. When developing the Linux-based driver for the display output controller it is necessary to consider many aspects associated with the kernel coding style, development patterns, quick changes in the API.

Linux graphics subsystem

Linux graphics subsystem (or graphics stack) is used for processing of graphic information. It consists of components, some of which reside in protected kernel space and some in user space. Components from the kernel space are used to interact with CPU registers, GPU registers, cache memory of various levels, RAM and GPU memory, etc. Memory allocated in this space is used by the kernel, kernel extensions, and modules to interact with devices. To enable graphics devices, DRM, KMS Linux components are often used. In the kernel space, the Direct Rendering Manager (DRM) module provides an API that user space programs use to send commands and data to the graphics kernel and configure display mode parameters [3]. DRM provides only the basic functionality that drivers can work with, and also provides user space with a certain minimum set of input-output system calls (ioctl) with standard hardware-independent functionality.

Some graphics hardware manufacturers use proprietary kernel modules (blob). Blob is a closed-source binary kernel module for open-source operating systems.

Linux provides several approaches for controlling the display of information. One of them is fbdev (framebuffer device). It is a framebuffer of a graphics device, allowing an application to access this device through

an interface, so the application does not need to directly access the low-level system. The fbdev interface does not provide for mode setting in the kernel. In turn, the framebuffer can be represented by a physical device or memory area for short-term storage of a given number of frames before sending them to a graphics output device.

KMS (kernel mode-setting) is a more modern approach. KMS allows to set parameters (screen resolution, color depth, refresh rate) for the display controller (DC) at the kernel level [4].

Approaches to implement 2D hardware acceleration

In most cases, the graphical user interface (GUI) used by Linux systems is built on top of an implementation of the X Windows System. There are several approaches to implement 2D hardware acceleration for Xorg using the GPU. One of them is the XAA architecture (XFree86 Acceleration Architecture) application. XAA manages offscreen memory by treating it as a large 2D area at a set number of bits per pixel (bpp). This architecture implements part of rendering operations, handling only cases where the destination picture is located in offscreen memory. This means that the source picture must be loaded each time into a free memory area before a 2D or 3D chip can handle it. Moreover, XAA only supports mask compositing when the source is a 1x1 repeating picture (a solid color) [5]. X.Org Server version 6.9 / 7.0 introduced the new EXA architecture as a replacement for XAA.

EXA provides an API for video drivers to implement 2D hardware acceleration. One of the key differences between EXA and XAA is that the former provides better integration with XRender (X Rendering Extension). Xorg often uses this extension to display anti-aliased fonts, fill a surface with a solid color, and implement compositing operations. Each acceleration operation in EXA is represented by three overloaded functions: Prepare <Action>, <Action> and Finish <Action>, where <Action> is the name of a specific operation. For some cases, DoneAction is used instead of FinishAction. Action can be called many times following a single call of PrepareAction for different surfaces. FinishAction is called after calling all Action. Basic 2D hardware acceleration operations are discussed below using the EXA architecture as an example.

The Solid operation fills an area with a solid color (RGBA). The Solid operation consists of three main functions: the first is the PrepareSolid function, which prepares the GPU for the solid color fill operation. The second is the Solid function used to perform filling. The FinishSolid function notifies the driver that the calling of a series of the Solid function has been completed, allowing it to restore the required GPU state.

The Copy operation copies a rectangular area (pixmap) in video memory from one bitmap to another. The Copy operation also has three main functions.

The Composite operation speeds up complex operations such as blending, scaling, and is optional for rendering. If the operation is not implemented in the driver, EXA performs the Composite operation using the pixman library.

The UploadToScreen operation copies an area from system memory to video memory.

The DownloadFromScreen operation copies an area from video memory to system memory.

The PrepareAccess function prepares the image for operations using the CPU.

This function moves the pixmap to system memory, copying it from video memory, which is inaccessible for the CPU.

The FinishAccess function is called when pixmap has been successfully accessed. It prepares the picture for use by the GPU.

EXA moves only those parts of the image that are necessary for the acceleration operation. For example, if A-> B-> C operations are performed and only B is accelerated with the GPU, A will be executed by the CPU, B will copy the pixmap from system memory to video memory, and the acceleration operation will be performed with the GPU. In turn, C will start moving the pixmap back to system memory and perform the third operation with the CPU. Since moving the pixmap creates significant overhead, the results will likely to be worse than those when all three operations are performed by the CPU. Therefore, the Composite operation can create a slowdown. To prevent a possible performance slowdown, it is necessary to create a profile of certain functions of the Composite operation to compile a complete list of all possible calls. For example, the sequence of these calls will vary depending on whether subpixel rendering is enabled or disabled when the font smoothing functions are called [6].

Preparing the environment for launching Xorg

X Windows System launch triggers an automatic search for the drivers installed in the system. If the X Window System fails to find the necessary hardware driver, it starts searching for the KMS driver first (x86-video-modesetting). If nothing is found, then X Windows System will search for the fbdev driver (xf86-video-fbdev). In the absence of the fbdev driver, the Xorg Windows System will use the standard vesa driver (xf86-video-vesa), which supports a wide range of x86 chipsets but fails to support 2D or 3D acceleration. To this day, the vesa driver is considered to be obsolete.

The xf86-video-fbdev driver is a hardware-independent graphics driver used by output devices. It allows applied software to access graphics devices via /dev/fb* interface. The fbdev driver uses the axillary fbdevhw module to access a framebuffer device [7].

The x86-video-modesetting (modesetting) driver is a hardware-independent X.Org driver, which works with any SoC that has the KMS driver operating at the kernel level. GLAMOR architecture is used for 2D hardware acceleration (provided that SoC has a graphics kernel supporting 3D acceleration), while DRI2 interface is used to address the library used for 3D graphics (for example, Mesa3D) [8].

Contemporary SoCs typically supports 2D and 3D hardware acceleration. Sometimes, a single SoC supports both 2D and 3D hardware acceleration. More often than not xf86-video-fbdev and x86-video-modesetting are used in the embedded systems when it is necessary to display graphic information without hardware acceleration. Linux kernel with a specific set of instructions (CONFIG_FB or CONFIG_DRM) need to be configured and auxiliary functions support needs to be enabled for the correct operation of fbdev and modesetting drivers. Also, it is necessary to describe the hardware components required for drivers operation. To do this, Linux offers the possibility to create a Device Tree Source (.dts) file. The advantage of the dts file is that a developer of the kernel modules does not need to recompile the kernel for various SoC platforms. The dtsi file containing the description of SoC architecture and blocks is represented as a node tree and is typically used to deliver information to the OS kernel. Device tree files can be divided into several parts in several files. Board-level files, i.e. dts, include the dtsi files, which describe SoC level [9].

After preparing the kernel and the dts file, it is necessary to change the Device section in the X server configuration file located in /etc/X11/xorg.conf, where the graphics driver used and its parameters are determined. The driver name, by which the user-space driver will be searched (for example, modesetting_drv.so) in the specified directory, is specified in the Driver line. The directory can be specified with the ModulePath parameter in the Files section. If no substandard path to the directory with modules has been specified when assembling Xorg, then /usr/lib/xorg/modules/drivers will be used by default. To assign an interface to access graphics devices, the following parameters can be used: fbdev (/dev/fb0) or kmsdev (/dev/dri/card0). Interface support is implemented in the graphics kernel driver. Optionally, such parameters as HWcursor (cursor hardware acceleration), AccelMethod (DDX (Device Dependent X) hardware acceleration enabling/disabling), etc. can be specified.

Results and discussion

When using SoCs in various projects, it is necessary to consider the specifics of their field of application. It is also worth noting that often SoCs are more expensive, so their use is not always justified. Let's consider the example of using the SoC as a single-board computer supporting 2D hardware acceleration. Single-board computers can be used for various purposes, for example, as a workstation for video surveillance, process monitoring, document management, etc. One of the performance criteria used to assess such systems is the GUI's responsiveness [10].

Xorg performance was tested with the x11perf utility version 1.5, which is included in the X applications package (x11-apps) and used for creating and displaying windows, displaying an existing set of windows (for example, maximizing the program window), dragging and dropping windows, displaying fonts, etc. [11]. Debian 8 was used as an operating system for testing. Testing was carried out in 5 modes, i.e. with 2D hardware acceleration and without. The summary performance diagrams with test results are presented below.

Figure 1 shows the results delivered by Xorg performance tests using the graphics kernel driver (the fbdev kernel-space driver and the 2D GPU + EXA user-space driver) and the fbdev driver (the fbdev kernel-space driver and the xf86-video-fbdev user-space driver). The results obtained with x11perf reflect what extent one driver is more efficient when performing a certain set of tests as compared to another in percentage terms. The diagram shows that the driver with 2D hardware acceleration support performs better (by 38.7 %) in Xorg performance tests than the fbdev driver [12]. The 2D driver delivers better results when running the tests with high-resolution images, for example, Copy 500x500 from window to pixmap, Copy 500x500 from pixmap to pixmap, Copy 500x500 from pixmap to window, Fill 300x300 aa trapezoid, GetImage XY 500x500 square, etc. In turn, the fbdev driver delivers better results with Copy 10x10 from window to pixmap, Copy 10x10 from pixmap to pixmap, Scroll 10x10 pixels, Dot, etc.

Figure 2 shows the results delivered by the graphics kernel driver (the fbdev kernel-space driver and the 2D GPU + EXA user-space driver) and the modesetting driver (the KMS kernel-space driver and the x86-video-modesetting user-space driver). In this case, the modesetting driver performs better than 2D GPU + EXA by 22 %. The modesetting significant advantages can be observed when performing the tests like Copy 10x10 from pixmap to window, Copy 10x10 from window to pixmap, Copy 10x10 from pixmap to pixmap, Resize unmapped window (4/16/25/50/75/100/200 kids), Moved unmapped window (4/16/25/50/75/100/200 kids), Fill 1x1 aa trapezoid.

To understand how EXA affects the test results, it has been decided to disable hardware interaction in 2D GPU + EXA user-space driver (kernel-space driver: fbdev) and compare the test results with those of modeset-

ting (the KMS kernel-space driver and the x86-video-modesetting user-space driver). The comparison results are shown in Figure 3.

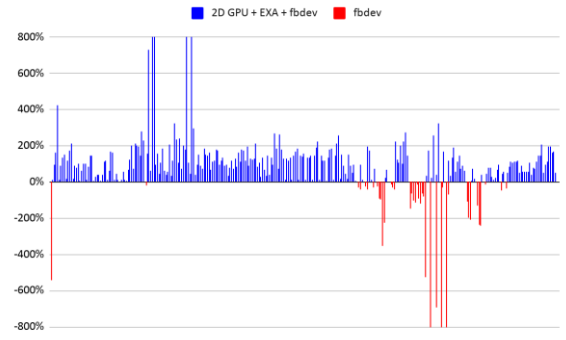

Fig. 1. Comparing the performance of 2D GPU + EXA + fbdev and fbdev

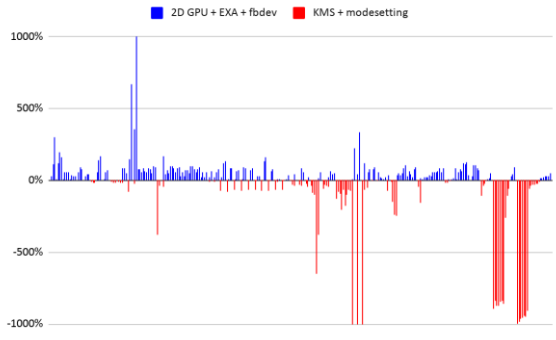

Fig. 2. Comparing the performance of 2D GPU + EXA + fbdev and KMS + modesetting

The data obtained lead to the concluded that the use of EXA significantly affects the test rate. For example, in the 500x500 tiled rectangle (216x208 tile) test, the 2D GPU + EXA driver with disabled hardware interaction (2D GPU + dummy EXA) performs better than modesetting by 37677 %. However, the graphics kernel driver (the fbdev kernel-space driver and the 2D GPU + EXA user-space driver) performs worse than modesetting by 17.9% in the same test.

For completeness of comparison, let's examine the effect of EXA on the performance when interacting with the DRM stack (Fig. 4). For EXA to work correctly with DRM, EXA_MIXED_PIXMAPS option has to be enabled in the 2D GPU + EXA driver, and the CreatePixmap2 function (instead of CreatePixmap), which allows to transfer to EXA information that the image has been resized by the driver to meet the hardware requirements, has to be used. The EXA_MIXED_PIXMAPS option allows to render those pixmaps with the CPU, rendering of which cannot be accelerated by the 2D driver. In certain situations, the absence of this option can lead to many visual artifacts in the form of unrendered areas when updating the framebuffer.

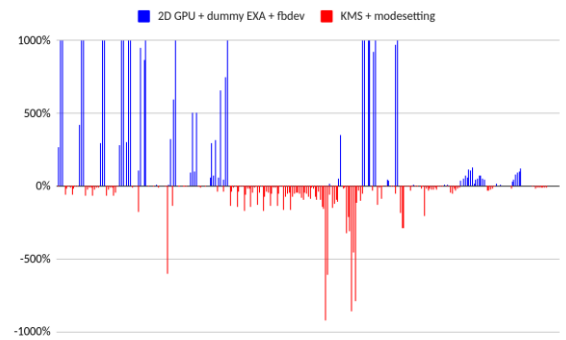

Fig. 3. Comparing the performance of 2D GPU + dummy EXA + fbdev and KMS + modesetting

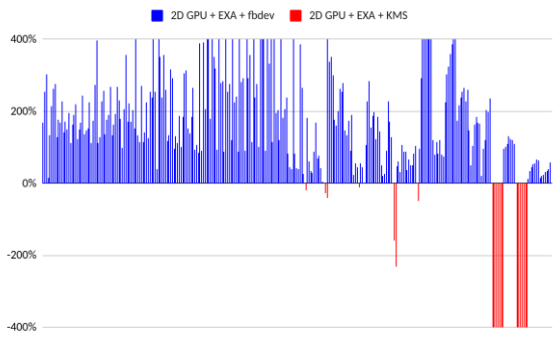

Fig. 4. Comparing the performance of 2D GPU + EXA + fbdev and 2D GPU + EXA + KMS

Figure 4 demonstrates the comparison of the performance of the kernel graphics drivers that use the fbdev kernel-space and the 2D GPU + EXA user-space driver, and the KMS kernel-space driver and the 2D GPU + EXA user-space driver. The test results presented in the diagram above lead to the conclusion that the use of EXA with the DRM stack has to be studied further. This is because in most cases scaling, pixmap copying, solid color filling, and other operations are processed by the CPU when using the DRM stack with the EXA_MIXED_PIXMAPS option.

The next stage required to test smooth running characteristics of the standard programs. Testing was conducted using LibreOffice Writer word processor as an example. The scrolling time of the open test document

with images, text and tables was measured in milliseconds (ms). The program response time was measured when typing and formatting the text (shown in the Table).

	KMS + modesetting	2D GPU + EXA + fbdev	fbdev
1 page scrolling (ms)	170	300	320
Text typing (ms)	30	98	100

When testing the performance with LibreOffice Writer, approx. 30 % decrease in the load of the graphics shell on the CPU was noted using 2D GPU + EXA + fbdev as compared to KMS + modesetting or fbdev drivers.

Conclusion

This article describes several methods that the X Windows System graphics system interacts with hardware.

The Xorg performance was tested with x11perf. To study the effect of the EXA API on the results, hardware interaction was disabled in the 2D GPU + EXA user-space driver (the fbdev kernel-space driver). Having compared the results with those delivered by modesetting (the KMS kernel-space driver and the x86-video-modesetting user-space driver), in can be concluded that the EXA API has a significant impact on the performance as the 2D GPU + EXA driver with disabled hardware interaction performs better only with a specific set of tests as compared to the modesetting driver.

Moreover, the test results lead to the conclusion that the use of 2D hardware acceleration with EXA is justified when high-resolution images are to be used or it is necessary to reduce the load on the CPU. It's worth noting that the use of EXA with the DRM stack has to be studied further, since, in most cases, scaling, copying, solid color filling, and other operations with the pixmap are implemented by the CPU when using the DRM stack with the EXA_MIXED_PIXMAPS option. In certain situations, the absence of this option can lead to many visual artifacts in the form of unrendered areas of the image when updating the framebuffer.

Acknowledgements: Publication is made as part of national assignment for SRISA RAS (fundamental scientific research 47 ГП) on the topic no. 0065-2019-0001 (AAAA-A19-119011790077-1).

References

1. Barr M., Massa A. *Programming Embedded Systems: With C and GNU Development Tools*. O'Reilly Media Publ., 2006, 328 p.
2. Bobkov S.G. *High-Performance Computing Systems*. Moscow, 2014. 296 p. (in Russ.).
3. *Kernel Mode Setting (KMS)*. Available at: https://www.kernel.org/doc/html/v4.15/gpu/drm-kms.html (accessed August 29, 2020).
4. Pugin K.V., Mamrosenko K.A., Giatsintov A.M., Visualization of graphic information in general-purpose operating systems. *Radioelectronics. Nanosystems. Information Technologies*, 2019, vol. 11, no. 2, pp. 217–224 (in Russ.). DOI: 10.17725/rensit.2019.11.217.
5. Anholt E. High Performance X Servers in the Kdrive Architecture. *Proc. USENIX Annual Techn. Conf., FREENIX Track*, 2004.
6. Marchesin S. *Linux Graphics Drivers: an Introduction*, 2012, 69 p. Available at: https://people.freedesktop.org/~marcheu/linuxgraphicsdrivers.pdf (accessed August 29, 2020).
7. Knorr G., Dänzer M., Uytterhoeven G. *Fbdev – Video Driver for Framebuffer Device*. Available at: https://www.x.org/archive/X11R6.8.0/doc/fbdev.4.html (accessed August 29, 2020).
8. Airlie D., *Modesetting – Video Driver for Framebuffer Device*, 2018. Available at: https://manpages.debian.org/stretch/xserver-xorg-core/modesetting.4.en.html (accessed August 29, 2020).
9. Efremov I.A., Mamrosenko K.A., Reshetnikov V.N., Methods of developing graphics subsystem drivers. *Software and Systems*, 2018, no. 3, pp. 425–429 (in Russ.). DOI: 10.15827/0236-235X.123.425-429.
10. Fan X. *Real-Time Embedded Systems: Design Principles and Engineering Practices*. Elsevier Science Publ., 2015, 1184 p.
11. McCormack J., Karlton Ph., Angebranndt S., Kent C., Packard K., Gill G. *X11perf – X11 Server Performance Test Program*. Available at: https://www.x.org/releases/X11R7.7/doc/man/man1/x11perf.1.xhtml (accessed August 29, 2020).
12. Madieu J. *Linux Device Drivers Development: Develop Customized Drivers for Embedded Linux*. Packt Publishing, 2017, 586 p.

УДК 004.92
DOI: 10.15827/2311-6749.20.3.2

Анализ методов взаимодействия графической системы X Windows System с аппаратным обеспечением

А.В. Родителев [1], *ведущий программист, rav@niisi.ras.ru*
К.А. Мамросенко [1], *к.т.н, заведующий отделом, mamrosenko_k@niisi.ras.ru*
А.М. Гиацинтов [1], *к.т.н., с.н.с., giatsintov@niisi.ras.ru*
В.Н. Решетников [1], *д.ф-м.н., профессор, главный научный сотрудник, rvn@niisi.ras.ru*

[1] *Центр визуализации и спутниковых информационных технологий НИИСИ РАН, Москва, 117218, Россия*

В статье описаны некоторые методы взаимодействия графической системы X Windows System с аппаратным обеспечением. На примере архитектуры EXA рассмотрены основные операции аппаратного 2D-ускорения.

Для исследования производительности было проведено тестирование Xorg с помощью утилиты x11perf версии 1.5. Тестирование проводилось в 5 режимах с использованием аппаратного 2D-ускорения и без. На основании результатов тестирования можно сделать вывод, что использование аппаратного 2D-ускорения с EXA оправдано в случаях, когда планируется использование изображений с высоким разрешением или есть необходимость снизить загрузку центрального процессора.

Тестирование производительности с отключенным взаимодействием с аппаратной частью (2D GPU + dummy EXA) показало, что использование EXA существенно влияет на скорость выполнения тестов, поскольку только на определенном наборе тестов драйвер 2D GPU + dummy EXA показывает лучший результат относительно драйвера modesetting.

Следует отметить, что использование EXA с DRM-стеком требует дополнительного исследования, так как при использовании DRM-стека с опцией EXA_MIXED_PIXMAPS в большинстве случаев операции масштабирования, копирования pixmap, заливки сплошным цветом и т.д. осуществляются на CPU. В определенных ситуациях отсутствие этой опции может привести к ряду графических артефактов в виде неперерисовывающихся областей изображения при обновлении кадрового буфера.

Ключевые слова: *EXA, установки режима ядра, X.Org Server, встроенная система, Linux.*

Благодарности. Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН (Фундаментальные исследования 47 ГП) по теме № 0065-2019-0001 «Математическое обеспечение и инструментальные средства для моделирования, проектирования и разработки элементов сложных технических систем, программных комплексов и телекоммуникационных сетей в различных проблемно-ориентированных областях» (АААА-А19-119011790077-1).

Литература

1. Barr M., Massa A. Programming Embedded Systems: With C and GNU Development Tools. O'Reilly Media Publ., 2006, 328 p.

2. Бобков С.А. Высокопроизводительные вычислительные системы. М.: Изд-во НИИСИ РАН, 2014. 296 с.

3. Kernel Mode Setting (KMS). URL: https://www.kernel.org/doc/html/v4.15/gpu/drm-kms.html (дата обращения 29.08.2020).

4. Пугин К.В., Мамросенко К.А., Гиацинтов А.М. Визуализация графической информации в операционных системах общего назначения // Радиоэлектроника. Наносистемы. Информационные технологии. 2019. Т. 11. № 2. С. 217–224. DOI: 10.17725/rensit.2019.11.217.

5. Anholt E. High Performance X Servers in the Kdrive Architecture. Proc. USENIX Annual Techn. Conf., FREENIX Track, 2004.

6. Marchesin S. Linux Graphics Drivers: an Introduction, 2012, 69 p. URL: https://people.freedesktop.org/~marcheu/linuxgraphicsdrivers.pdf (дата обращения 29.08.2020).

7. Knorr G., Dänzer M., Uytterhoeven G. Fbdev – Video Driver for Framebuffer Device. URL: https://www.x.org/archive/X11R6.8.0/doc/fbdev.4.html (дата обращения 29.08.2020).

8. Airlie D., Modesetting – Video Driver for Framebuffer Device, 2018. URL: https://manpages.debian.org/stretch/xserver-xorg-core/modesetting.4.en.html (дата обращения 29.08.2020).

9. Ефремов И.А., Решетников В.Н., Мамросенко К.А. Методы разработки драйверов графической подсистемы // Программные продукты и системы. 2018. № 3. С. 425–429.

10. Fan X. Real-Time Embedded Systems: Design Principles and Engineering Practices. Elsevier Science Publ., 2015, 1184 p.

11. McCormack J., Karlton Ph., Angebranndt S., Kent C., Packard K., Gill G. X11perf – X11 server performance test program. URL: https://www.x.org/releases/X11R7.7/doc/man/man1/x11perf.1.xhtml (дата обращения 29.08.2020).

12. Madieu J. Linux Device Drivers Development: Develop Customized Drivers for Embedded Linux. Packt Publishing, 2017, 586 p.