Academicia Globe:
Inderscience Research

ISSN: 2776-1010 Volume 2, Issue 10, Oct, 2021

APPROACHES TO PROVIDING DATA VISUALIZATION ON DEVICES USING MODERN REAL-TIME OPERATING SYSTEMS

Zainab Qahtan Mohammed Abass
Department of Computer Science, College of Basic Education, University of Diyala
Email: mm4114272@gmail.com

Enas Mahmood Jassim Hamoody
Department of Computer Science, college of Basic Education, University of Diyala
Email: enasmahmod79@gmail.com

Abstract
The paper considers various approaches to developing display controller driver software for embedded systems, which usually use System-on-chip (SOC) solutions and real-time operating systems (RTOS). It also describes the main principles and design decisions of the chosen RTOS, such as portability, flexible scheduling, responsiveness, etc., as well as used standards (C language, POSIX 1003.1, ARINC 653). The authors list general operating principles of the display controller hardware including support for several displays and overlays that can be used for displaying multiple video streams on one screen or achieving the effect similar to chroma keying. The proposed method for developing display controller drivers defines the main parameters of the display controller hardware and the steps necessary to show an image on a screen correctly. The meth-od also takes into account the features related to using hardware interrupts and estimating the frequen-cy required for display controller to show an image on a screen correctly with the defined screen mode. Contrary to the known methods, the proposed method takes into account various features of the many real-time operating systems: lack of dedicated API to interact with graphics hardware and resources, seamless access to hardware registers from user space and so on. The paper considers several ap-proaches to debugging software on the target hardware as well as using prototyping systems based on FPGA. Prototyping systems usually introduce additional challenges to debugging, such as low simula-tion speed. The proposed method was tested during the development of the display controller driver for the home made RTOS..

Keywords: ARINC , posix , embedded systems , x window system , the output controller at the screen the driver , real-time operating systems , visualization.

INTRODUCTION

Embedded Systems (ES) are gaining in popularity. They have low heat emission and smaller dimensions, which allows using small-sized cooling systems or even doing without them. ES are widely used in such areas as aviation, robotics, astronautics, mechanical engineering, medicine, Internet of Things (IoT), etc. It is possible to highlight the use of such systems in analytical instruments with a built-in screen, for example, an oscilloscope, an X-ray diffractometer, etc. Due to their small size and

ISSN: 2776-1010 Volume 2, Issue 10, Oct, 2021

power consumption, the computing capabilities of embedded systems are quite small, which imposes a number of restrictions on their use.

As a rule, single-chip systems, or systems on a chip (SoC, System on chip, SoC) are used in embedded systems. A number of components can be located inside a single die, such as a CPU, GPU, memory controller, etc. SoCs with integrated graphics are widely used in mobile devices (smartphones) or mini-computers (for example, Raspberry Pi).

In the design of advanced systems-on-a-chip, a component can be used that provides output of graphical information to display devices such as monitors or embedded screens. Typically, this component is a Display Controller (DC). Depending on the tasks of using the SoC, the functionality of the DC built into it may differ.

ES requirements determine the use of different OS [2]. Therefore, it is necessary to develop approaches to the display of graphical information, as well as to the design of drivers for various operating systems, including real-time (RT) [3].

Currently, there are a number of embedded systems for which it is important to respond in a timely manner to emerging events. As a rule, RT OS is used for such purposes. Within the framework of this article, the domestic OS RT is considered [4], which is based on the following general principles:

- Use of standards;
- Mobility;
- Development of tools for logging diagnostics and error handling;
- Flexible planning tools;
- Using an object-oriented approach;
- Manageability (in particular, the availability of configuration tools);
- Availability of cross-development and debugging tools for custom applications;
- The presence of a significant number of environment packages for creating graphical applications, databases, cartographic systems.

When developing this OS, the following international standards and specifications were used:

- The C standard, which describes the C language and libraries;
- Standard for mobile OS (software interface) POSIX 1003.1;
- The ARINC 653 specification, which defines the APEX (APplication EXecutive) interface between the OS of the target module and application programs [5].

Designing drivers for RT OS differs from designing for other operating systems. You can consider the differences using the example of Linux OS - one of the actively developing and used open source projects. Linux has a number of APIs to better interact with devices and resources. For example, the API, implemented by the Linux OS subsystem, called the Direct Rendering Manager (DRM) [6], is responsible for the interaction of the OS with the GPU, video memory, etc. It simplifies interaction with memory, registers, GPU interrupts. In the case of the RT OS, such a subsystem may not exist; therefore, access to physical memory and interrupt control can occur directly. It is also necessary to take into account such a significant feature of the RT OS when designing drivers, as a direct reference to the address space of the device registers.

ISSN: 2776-1010 Volume 2, Issue 10, Oct, 2021

In contrast to the Linux OS, the RV OS used is of the microkernel type [5]. In this type of OS, device drivers run as separate processes in user space and can be represented as a separate library, while in Linux, a device driver is usually represented as a kernel module, which must also be taken into account when designing.

This paper proposes a new method for designing display controller drivers, in contrast to the known solutions, taking into account the above features of the RT OS.

Analysis of the Results of Previous Work

The article [7] analyzes the development of drivers for Windows and Linux, but not for RT. In [8], the authors describe the operation of graphical applications on various RT operating systems and the problems that one has to face when developing such applications, but the issues related to the output of the generated image to the information display device (IDD) are not considered. The article [9] describes the application of the RT OS in the tasks of capturing a video signal, approaches to receiving, processing and displaying it on the screen. However, a small panel used for microcontrollers is used as a screen, which is very different from using a separate built-in display controller and standard monitors. The work [10] provides data on the methods of using VxWorks RT OS in flight displays, but there is no information about low-level operations for displaying a graphic image. A large number of studies related to the use of RT operating systems indicate the relevance of the development of drivers for such systems.

Research Methods and Materials

Typically, a DC is a device that reads data from a frame buffer, processes it if required, and passes it on to display devices. The location of the frame buffer may vary from system to system. In the case of desktop personal computers with discrete graphics, the frame buffer can be located in the GPU video memory. In the case of SoC, in most cases, an integrated graphics adapter is used that does not have its own RAM, therefore, such systems use RAM common to the CPU and GPU. Therefore, to store the frame buffer, shared RAM is used, part of which is pre-reserved for the needs of the GPU.

Many DCs support the use of overlay technology, which is one of the types of surfaces (planes) [11]. Surfaces are represented as rectangular areas with their own sizes, positions, formats. The number of surfaces used is usually limited by DC capabilities, for example, some controllers can support up to three surfaces. Surfaces can be primary, secondary framebuffers, cursor framebuffer, etc. The primary buffer is the main one, and the secondary buffer is used when working with overlays.

Overlay technology can be applied in different ways. For example, there is a main framebuffer, which can cover the entire field of view, and a secondary framebuffer. The DC reads the data from the backbuffer and overlays the backbuffer image on top of the main buffer using the specified blending function. The blending function determines the actions to be performed on each pixel of the buffers used. For example, the blending function allows you to set a color that will be considered transparent when overlaid. The result is a chroma keying-like effect that is widely used in various fields.

Academicia Globe:
Inderscience Research

ISSN: 2776-1010 Volume 2, Issue 10, Oct, 2021

Another example of using such functions is displaying video streams. For example, there is a technical video capture device or a video decoder from which we receive frames. These frames will be displayed on the specified area of the primary frame buffer. The displayed area of the video can be smaller than the primary buffer, and its size and offset can be controlled by the DC.
A number of DCs have hardware support for two or more displays, which, for example, allows the device to operate in dual display mode. This mode can be used to clone an image from one screen to another, or to expand the rendering area on a second screen. The first case can be used in the tasks of presentations or additional monitoring, the second - for example, for VR devices.

Research Results and their Discussion

The functioning of a DC is usually provided by the set of parameters it contains. These include, for example, the addresses of frame buffers, their properties (size, resolution), refresh rates, etc. The set of these characteristics may vary depending on the model of the DC used. In case of significant differences in DC for each device, it is advisable to create a separate driver. The following describes the approach to programming DC, as well as the differences in the devices.
Before programming the DC to display the image on the screen, it is necessary to determine the operating frequency of the controller. The synthesizer (PLL, Phase-locked loop, PLL) is responsible for this, which, depending on the microcircuits, can consist of several stages. The microcircuits used use a two-stage synthesizer. For each stage, the coefficient values are calculated based on the specified parameters (input frequency divider, synthesizer mode, etc.). These values are written to the synthesizer registers, after which DC programming is performed.
The first step in programming a DC is to reset its state. This is necessary for further DC programming correctness. Typically, the processing of the clock, clock, and data frequencies is disabled.
The next step is to set the size of the visible area, as well as the vertical and horizontal refresh rates. Setting these parameters affects the correctness of information display. If the update parameters are incorrect, as a result, artifacts can be observed in the form of displacement of the visible area or the absence of an image.
Next, the parameters are set that are responsible for working with the frame buffer, which contains the data for display. The following parameters can be set for DC: frame buffer address, its size, format, DPI and step. The DPI parameter is responsible for the arrangement of the colors of the output bus. The step parameter determines the size of one line in bytes. Typically, the step value is equal to the framebuffer width times the color depth (bits per pixel, bpp). Some DCs may not have DPI settings and buffer size.
Optionally, parameters related to hardware cursor, gamma, dithering can be specified, provided that the device supports setting these parameters. Dithering allows you to set the degree of color intensity of neighboring pixels in case of noticeable jumps in intensity.
Another parameter that can be set during DC programming is interrupt handling. In the case of DC, interrupts are needed to handle, for example, a hardware cursor. In this case, interrupts will give a signal to update only the cursor plane, and not the entire frame buffer area. If interrupts are to be used

ISSN: 2776-1010 Volume 2, Issue 10, Oct, 2021

on the system (for example, Linux DRM requires the use of interrupts for display controllers), then interrupt handling must be enabled. It is also necessary to read the DC interrupt register in the interrupt handler so that the controller registers the interrupt processing and can continue to work. In the case of the RT OS, you should separately enable the necessary interrupt and register a handler for it in the kernel space using OS functions. The handler is a function, which reads the register to activate the DC interrupt. When a device driver is implemented as a static library for a POSIX process, there is often a significant amount of interrupt handling required. At the same time, the RT OS requires interrupt handling in the kernel module. To accommodate this limitation, the POSIX process library creates an interrupt thread waiting for the semaphore signal. In the kernel module, a function is created that raises the semaphore when an interrupt arrives. Given that semaphores are available to various processes and the kernel, a raised semaphore will start the interrupt thread in the driver library. At the same time, the RT OS requires interrupt handling in the kernel module. To accommodate this limitation, the POSIX process library creates an interrupt thread waiting for the semaphore signal. In the kernel module, a function is created that raises the semaphore when an interrupt arrives. Given that semaphores are available to various processes and the kernel, a raised semaphore will start the interrupt thread in the driver library. At the same time, the RT OS requires interrupt handling in the kernel module. To accommodate this limitation, the POSIX process library creates an interrupt thread waiting for the semaphore signal. In the kernel module, a function is created that raises the semaphore when an interrupt arrives. Given that semaphores are available to various processes and the kernel, a raised semaphore will start the interrupt thread in the driver library.

The last stage in programming is the reverse operation of the first stage - setting the DC operating state: the processing of synchronization frequencies, clock generator and data is turned on.

The described stages of DC programming are the main ones when developing a driver. Depending on the hardware capabilities of the DC, the functionality of interacting with overlays is additionally implemented.

The DC driver consists of two parts - a static library for the POSIX process and a kernel module. The main programming of the device is done by the library for the POSIX process, while the kernel module is used to obtain the virtual address of the DC register space, as well as to allocate memory to the frame buffer used to display information on the screen. The interaction of the library for the POSIX process and the kernel module in the RT OS is implemented using ioctl with parameters defined by the kernel module. Based on this, a function was implemented that, using ioctl, obtains the address of the DC register space for direct access to them.

Due to the lack of a specialized API in the RT OS for interacting with information display devices, its own API was developed, in which a separate function was created for each stage of DC programming. Most of the display controller parameters are set through separate registers. The base address of the register space and its offset from the base are used to set the value of a particular register. Considering this feature, in the developed API for each function, the first parameter is a pointer to the address of the device register space, the rest of the parameters depend on the purpose of a specific function.

ISSN: 2776-1010 Volume 2, Issue 10, Oct, 2021

It is worth noting a point related to byte ordering. The task was to ensure the operability of the driver using a different byte order: from high to low (big-endian) and from low to high (little-endian). Taking into account that the used display controller was designed to work in little-endian systems, a function was added that changes the byte order of the written value in the case of using an OS with big-endian.

In the process of software development, the stage of its debugging is important. If a physical device or a prototype already exists, you can resort to well-known software debugging tools such as GDB. Debugging can be done remotely from the tooling machine to the target over the network. In most cases, remote debugging is done via a network connection, and occasionally via a UART interface. If you cannot use network debugging, one solution may be to embed debug output in the source code for output to the console. This method is common when debugging the kernel-part of the OS, since the use of the debugger in the kernel in most cases is significantly limited.

However, it is not always possible to debug a software component if the SoC, on which you want to debug the component, is in active development. In such cases, FPGA-based prototyping systems are used, for example, Protium or Altera, on which microprocessor emulation is carried out. However, such systems have their own peculiarities, for example, low speed of command execution or the inability to model all complex-functional blocks. The use of a debugger together with a prototyping system is also impractical in some cases due to the low speed of command execution or the instability of the prototype. In addition, the network interface is not always present in the prototype.

Approbation of the described approaches was carried out in the process of developing the DC driver, which is part of the X Window System server [13], for the developed domestic RT OS. The developed driver allows performing a number of new operations, unlike the previous solution, for example, displaying images on several screens with support for various video interfaces, such as DVI and LVDS, as well as programming overlays.

Conclusion

Providing visualization of graphical information in embedded systems using promising single-chip systems is an important task. One of the main components for rendering is the display controller. In this paper, a method for designing display controller drivers was presented, taking into account the peculiarities of work in the RT OS. The main parameters of the controller have been determined, which must be set for its correct operation. Parameters are also provided to enable specialized functionality such as overlays or dithering. Comparison of approaches to programming DC in different OS is carried out on the example of Linux and RT OS.

The approach described in the article was tested in the development of the DC driver - a separate component for the RT OS as part of the X Window System server. As a direction for further work, we can consider the study and development of a graphics subsystem for RT OS.

ISSN: 2776-1010 Volume 2, Issue 10, Oct, 2021

REFERENCES

1- Hobbs C. Embedded Software Development for Safety-Critical Systems . 2019, 364 p. DOI: 10.1201 / b18965 .
2- Bertolotti I., Manduchi G. Real-Time Embedded Systems: Open-Source Operating Systems Perspective. USA, Florida, Boca Raton, CRC Press publ., 2012, 534 p.
3- Drozdov A. Yu., Fonin Yu.N., Perov MN, Vedishcheva TS, Novoselova Yu.K. An approach to cross-platform drivers development. Proc. Intern. Conf. EnT , 2015, pp. 54-57.
4- Lefebvre Y. An embedded platform-agnostic solution to deploy graphical applications. SAE Technical Paper Series , 2011, no. 2011-01-2551.
5- Palatty JJ, Edireswarapu SPC, Sivraj P. Performance analysis of FreeRTOS based video capture system. Proc. III ICECA , 2019, pp. 595-599.
6- Wu Y., Xuejun Z., Huaxian L., Xiangmin G. Design and realization of Display Control software in Integrated Avionic system for General Aviation based on the VxWorks. Proc. XIII WCICA , 2018, pp. 1295-1299.
7- Barry P., Crowley P. Modern Embedded Computing Designing Connected, Pervasive, Media-Rich Systems . Morgan Kaufmann publ., 2012, 544 p.