УДК 004.454
DOI: 10.15827/0236-235X.135.433-439
Дата подачи статьи: 17.06.21
2021. Т. 34. № 3. С. 433–439

Подходы к обеспечению визуализации данных на устройствах с использованием современных операционных систем реального времени

П.С. Баженов [1], *ведущий программист, bps@niisi.ras.ru*
А.М. Гиацинтов [1], *к.т.н., старший научный сотрудник, giatsintov@niisi.ras.ru*
К.А. Мамросенко [1], *к.т.н., руководитель Центра, mamrosenko_k@niisi.ras.ru*

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

В работе рассматриваются подходы к проектированию драйвера контроллера вывода на экран для встраиваемых систем, в которых, как правило, применяются системы на кристалле и современные операционные системы реального времени. Описаны общие принципы используемой операционной системы реального времени, такие как мобильность, гибкие средства планирования, управляемость и т.д., а также применение международных стандартов (стандарт C, POSIX 1003.1, ARINC 653). Приведены общие принципы работы контроллеров вывода на экран, в том числе реализация оверлеев, предназначенных для отображения видеопотоков на одном экране либо достижения эффекта, подобного хромакеингу, возможности аппаратной поддержки двух и более дисплеев.

В предлагаемом методе проектирования драйверов определяются основные параметры контроллера вывода на экран, порядок задания данных параметров для корректного вывода изображения на экран. Также приняты во внимание особенности, связанные с использованием функциональности прерываний и определением рабочей частоты контроллера. В отличие от известных решений в методе учитываются особенности ряда операционных систем реального времени: отсутствие специализированного API для взаимодействия с графическими устройствами и ресурсами, прямое обращение к регистрам устройств из пользовательского пространства. Рассмотрены подходы к отладке ПО на устройстве, а также с использованием систем прототипирования, основанных на ПЛИС, имеющих ряд особенностей, в том числе низкую скорость выполнения операций.

Метод апробирован при разработке драйвера контроллера вывода на экран – отдельного компонента для отечественной операционной системы реального времени в составе сервера X Window System.

Ключевые слова: *визуализация, операционные системы реального времени, драйвер, контроллер вывода на экран, X Window System, встраиваемые системы, POSIX, ARINC.*

Встраиваемые системы (Embedded System, ES) приобретают все большую популярность. Они обладают низким тепловыделением и меньшими размерами, что позволяет использовать малогабаритные системы охлаждения или вовсе обходиться без них. ES находят широкое применение в таких областях, как авиация, робототехника, космонавтика, машиностроение, медицина, интернет вещей (IoT) [1] и т.д. Можно выделить использование таких систем в аналитических приборах со встроенным экраном, например, осциллограф, рентгеновский дифрактометр и проч. В силу малых размера и энергоемкости вычислительные возможности встраиваемых систем довольно небольшие, что накладывает ряд ограничений на их использование.

Как правило, во встраиваемых системах применяются однокристальные системы, или *системы на кристалле* (СнК, System on chip, SoC). Внутри одного кристалла могут находиться ряд компонентов, таких как CPU, GPU, контроллер памяти и т.д. СнК с встроенной графикой широко применяются в мобильных устройствах (смартфонах) или в мини-компьютерах (например Raspberry Pi).

При проектировании перспективных систем на кристалле может использоваться компонент, обеспечивающий вывод графической информации на устройства отображения, например, мониторы или встраиваемые экраны. Как правило, таким компонентом является контроллер вывода на экран (Display Controller, DC). В зависимости от задач использования SoC функциональность встроенного в него DC может отличаться.

Требования к ES обусловливают применение различных ОС [2]. Поэтому необходима выработка подходов к отображению графической информации, а также к проектированию

драйверов для различных ОС, в том числе *реального времени* (РВ) [3].

В настоящее время существует ряд встраиваемых систем, для которых важно своевременно реагировать на возникающие события. Как правило, для таких целей используются ОС РВ. В рамках данной статьи рассматривается отечественная ОС РВ [4], которая базируется на следующих общих принципах:

– использование стандартов;

– мобильность;

– развитие средств протоколирования диагностики и обработки ошибок;

– гибкие средства планирования;

– использование объектно-ориентированного подхода;

– управляемость (в частности, наличие средств конфигурирования);

– наличие кросс-средств разработки и отладки пользовательских приложений;

– наличие значительного числа пакетов окружения для создания графических приложений, БД, картографических систем.

При разработке данной ОС использованы следующие международные стандарты и спецификации:

– стандарт С, описывающий язык и библиотеки языка С;

– стандарт на мобильные ОС (программный интерфейс) POSIX 1003.1;

– спецификация ARINC 653, определяющая интерфейс APEX (APplication EXecutive) между ОС целевого модуля и прикладными программами [5].

Проектирование драйверов для ОС РВ отличается от проектирования для иных ОС. Можно рассмотреть различия на примере ОС Linux – одного из активно развивающихся и применяемых проектов с открытым исходным кодом. Linux имеет ряд API для более удобного взаимодействия с устройствами и ресурсами. Например, API, реализованный подсистемой ОС Linux, называемой Direct Rendering Manager (DRM) [6], отвечает за взаимодействие ОС с GPU, видеопамятью и т.д. С ее помощью упрощается взаимодействие с памятью, регистрами, прерываниями графического процессора. В случае ОС РВ такой подсистемы может не быть, поэтому доступ к физической памяти и управление прерываниями могут происходить напрямую. Необходимо также учесть такую значимую при проектировании драйверов особенность ОС РВ, как прямое обращение к адресному пространству регистров устройства. Может потребоваться разработка отдельной подсистемы управления, что является отдельной большой задачей.

В отличие от ОС Linux используемая ОС РВ относится к микроядерному типу [5]. В ОС такого типа драйверы устройств работают как отдельные процессы в пользовательском пространстве и могут представляться в виде отдельной библиотеки, в то время как в ОС Linux драйвер устройства, как правило, представляется в виде модуля ядра, что также нужно учитывать при проектировании.

В данной работе предлагается новый метод проектирования драйверов контроллера вывода на экран, в отличие от известных решений учитывающий указанные выше особенности ОС РВ.

Анализ результатов предшествующих работ

В статье [7] проведен анализ разработки драйверов для ОС Windows и Linux, но не для ОС РВ. В [8] авторы описывают работу графических приложений на различных ОС РВ и проблемы, с которыми приходится сталкиваться при разработке подобных приложений, однако не рассмотрены вопросы, связанные с выводом сгенерированного изображения на *устройство отображения информации* (УОИ). В статье [9] описано применение ОС РВ в задачах захвата видеосигнала, приводятся подходы к получению, обработке и выводу на экран. Однако в качестве экрана используется небольшая панель, применяемая для микроконтроллеров, что в значительной степени отличается от использования отдельного встроенного контроллера вывода на экран и стандартных мониторов. В работе [10] приводятся данные о методах применения ОС РВ VxWorks в пилотажных дисплеях, но нет информации о низкоуровневых операциях по выводу графического изображения. Большое количество исследований, связанных с использованием ОС РВ, указывают на актуальность разработки драйверов для таких систем.

Методы и материалы исследования

Как правило, DC является устройством, которое считывает данные с кадрового буфера (frame buffer), выполняет их обработку, если требуется, и передает дальше на устройства отображения информации. В разных системах расположение кадрового буфера может отли-

чаться. В случае настольных персональных компьютеров с дискретной графикой кадровый буфер может находиться в видеопамяти GPU. В случае SoC в большинстве случаев применяется встроенный графический адаптер, не имеющий собственной оперативной памяти, поэтому в таких системах используется общая для CPU и GPU оперативная память. Следовательно, для хранения кадрового буфера используется общая оперативная память, часть которой предварительно резервируется для нужд GPU.

Многие DC поддерживают использование технологии оверлеев (overlay), являющихся одним из видов поверхностей (planes) [11]. Поверхности представляются как прямоугольные области со своими размерами, позициями, форматами. Количество используемых поверхностей, как правило, ограничено возможностями DC, например, некоторые контроллеры могут поддерживать до трех поверхностей. В роли поверхностей могут выступать первичные, вторичные кадровые буферы, кадровый буфер курсора и т.д. Первичный буфер является основным, а вторичный вспомогательным, используемым при работе с оверлеями.

Технология оверлея может применяться по-разному. Например, имеются основной кадровый буфер, который может покрывать всю область видимости, и вторичный. DC считывает данные из вторичного буфера и накладывает изображение из вторичного буфера поверх основного с использованием заданной функции смешивания. Функция смешивания определяет действия, выполняемые над каждым пикселем используемых буферов. Так, функция смешивания позволяет задать цвет, который при наложении будет считаться прозрачным. В итоге получается эффект, подобный хромакеингу (chroma keying), который широко используется в различных областях.

Другой пример использования таких функций – отображение видеопотоков. Например, имеется техническое устройство захвата видео или видеодекодер, с которых получаем кадры. Данные кадры будут отображаться на указанной области первичного кадрового буфера. Отображаемая область видео может быть меньше первичного буфера, ее размеры и смещение могут управляться DC.

Ряд DC имеют аппаратную поддержку двух и более дисплеев, что, например, позволяет устройству работать в режиме dual display. Этот режим можно применять для клонирования изображения с одного экрана на другой или для расширения области визуализации на втором экране. Первый случай может быть использован в задачах презентаций или дополнительного мониторинга, второй – например, для VR-устройств.

Результаты исследований и их обсуждение

Функционирование DC, как правило, обеспечивает имеющийся в нем набор параметров. К ним относятся, например, адреса кадровых буферов, их свойства (размер, разрешение), значения частот обновления и т.д. Набор данных характеристик может варьироваться в зависимости от модели используемого DC. В случае значительных различий DC под каждое устройство целесообразно создавать отдельный драйвер. Далее описываются подход к программированию DC, а также различия в устройствах.

Перед программированием DC для вывода изображения на экран необходимо определить рабочую частоту контроллера. За это отвечает синтезатор (PLL, Phase-locked loop, ФАПЧ), который в зависимости от микросхем может состоять из нескольких ступеней. В применяемых микросхемах используется двухступенчатый синтезатор. Для каждой ступени рассчитываются значения коэффициентов на основе задаваемых параметров (входной делитель частоты, режим синтезатора и т.д.). Эти значения записываются в регистры синтезатора, после чего выполняется программирование DC.

На первом этапе программирования DC осуществляется сброс его состояния. Это необходимо для дальнейшей корректности программирования DC. Как правило, выполняется отключение обработки частот синхронизации, тактового генератора и данных.

На следующем шаге задаются размеры видимой области, а также частоты обновления по вертикали и горизонтали. Задание данных параметров влияет на корректность отображения информации. Если параметры обновления некорректны, в итоге могут наблюдаться артефакты в виде смещения видимой области либо отсутствие изображения.

Далее задаются параметры, отвечающие за работу с кадровым буфером, который содержит в себе данные для отображения. Для DC могут задаваться следующие параметры: адрес кадрового буфера, его размер, формат, DPI и шаг. Параметр DPI отвечает за расположение цветов выходной шины. Параметр шага определяет

размер одной строки в байтах. Как правило, значение шага равно ширине кадрового буфера, умноженной на значение глубины цвета (bits per pixel, bpp). В некоторых DC параметры DPI и размер буфера могут отсутствовать.

Опционально могут указываться параметры, связанные с аппаратным курсором, гаммой, дитерингом, при условии, что устройство поддерживает настройку данных параметров. Дитеринг позволяет задать степень интенсивности цвета соседних пикселей в случае наблюдения заметных скачков интенсивности.

Еще одним параметром, который может задаваться при программировании DC, является обработка прерываний. В случае DC прерывания необходимы для обработки, например, аппаратного курсора. В таком случае прерывания будут давать сигнал на обновления только плоскости курсора, а не всей области кадрового буфера. Если в системе должны использоваться прерывания (например, Linux DRM требует использования прерываний для контроллеров вывода на экран), то необходимо включить обработку прерываний. Также в обработчике прерываний необходимо выполнять считывание регистра прерывания DC, чтобы контроллер зарегистрировал обработку прерывания и мог продолжить работу. В случае с ОС РВ следует отдельно включать необходимое прерывание и регистрировать для него обработчик в пространстве ядра, используя функции ОС. Обработчик представляет собой функцию, выполняющую считывание регистра для активации прерывания от DC. Когда драйвер устройства реализован в виде статической библиотеки для POSIX-процесса, зачастую требуется выполнять значительное количество действий при обработке прерываний. В то же время ОС РВ требует обработки прерываний в модуле ядра. Для учета этого ограничения в библиотеке для POSIX-процесса создается поток обработки прерываний, ожидающий сигнала семафора. В модуле ядра создается функция, поднимающая семафор по приходу прерывания. С учетом того, что семафоры доступны различным процессам и ядру, поднятый семафор запустит поток обработки прерываний в библиотеке драйвера.

Последним этапом при программировании является обратная операция первого этапа – установка рабочего состояния DC: происходит включение обработки частот синхронизации, тактового генератора и данных.

Описанные этапы программирования DC являются основными при разработке драйвера. В зависимости от аппаратных возможностей DC дополнительно реализуется функциональность взаимодействия с оверлеями.

Драйвер DC состоит из двух частей – статической библиотеки для POSIX-процесса и модуля ядра. Основное программирование устройства выполняется библиотекой для POSIX-процесса, в то время как модуль ядра используется для получения виртуального адреса пространства регистров DC, а также выделения памяти кадровому буферу, используемому для отображения информации на экране. Взаимодействие библиотеки для POSIX-процесса и модуля ядра в ОС РВ реализуется при помощи ioctl с параметрами, определяемыми модулем ядра. На основе этого была реализована функция, которая с помощью ioctl получает адрес пространства регистров DC для прямого обращения к ним.

В связи с отсутствием в ОС РВ специализированного API для взаимодействия с устройствами отображения информации был разработан свой API, в котором для каждого этапа программирования DC создана отдельная функция. Большинство параметров контроллера вывода на экран задаются через отдельные регистры. Для задания значения конкретного регистра используются базовый адрес пространства регистров и его смещение относительно базового. Учитывая эту особенность, в разработанном API у каждой функции первым параметром является указатель на адрес пространства регистров устройства, остальные параметры зависят от назначения конкретной функции.

Стоит отметить момент, связанный с порядком байтов. Стояла задача по обеспечению работоспособности драйвера с использованием различного порядка байтов: от старшего к младшему (big-endian) и от младшего к старшему (little-endian). С учетом того, что применяемый контроллер вывода на экран спроектирован для работы в системах little-endian, была добавлена функция, изменяющая порядок байтов записываемого значения в случае использования ОС с big-endian.

В процессе разработки ПО немаловажен этап его отладки. Если физическое устройство или его прототип уже существует, можно прибегнуть к широко известным инструментам для отладки ПО, например GDB. Отладка может производиться удаленно с инструментальной машины на целевой по сети. В большинстве случаев удаленная отладка производится через сетевое подключение, изредка – через

UART-интерфейс. При невозможности использовать отладку по сети одним из решений может быть внедрение вывода отладочной информации в исходный текст для вывода в консоль. Такой метод распространен при отладке kernel-части ОС, так как использование отладчика в ядре в большинстве случаев существенно ограничено.

Однако не всегда есть возможность выполнить отладку программного компонента, если СнК, на котором требуется отладить компонент, находится в активной разработке. В таких случаях используются системы прототипирования на основе ПЛИС, например, Protium или Altera, на которых осуществляется эмуляция микропроцессора [12]. Однако у таких систем есть свои особенности, например, низкая скорость выполнения команд или невозможность смоделировать все сложно-функциональные блоки. Использование отладчика вместе с системой прототипирования также в некоторых случаях нецелесообразно из-за низкой скорости выполнения команд или нестабильности прототипа. К тому же не всегда в прототипе присутствует сетевой интерфейс.

Апробация описанных подходов проведена в процессе разработки драйвера DC, входящего в состав сервера X Window System [13], для разрабатываемой отечественной ОС РВ. Разработанный драйвер позволяет выполнить ряд новых операций в отличие от предыдущего решения, например, вывод изображения на несколько экранов с поддержкой различных видеоинтерфейсов, таких как DVI и LVDS, а также программирование оверлеев.

Заключение

Обеспечение визуализации графической информации в встраиваемых системах с применением перспективных однокристальных систем – важная задача. Одним из основных компонентов, обеспечивающих визуализацию, является контроллер вывода на экран. В данной работе был представлен метод проектирования драйверов контроллера вывода на экран, учитывающий особенности работы в ОС РВ. Определены основные параметры контроллера, которые требуется задавать для его корректной работы. Также приведены параметры, позволяющие задействовать специализированную функциональность, например, оверлеи или дитеринг. Проведено сравнение подходов к программированию DC в различных ОС на примере Linux и ОС РВ.

Подход, описанный в статье, был апробирован при разработке драйвера DC – отдельного компонента для ОС РВ в составе сервера X Window System. В качестве направления дальнейшей работы можно рассмотреть исследование и разработку графической подсистемы для ОС РВ.

Работа выполнена в рамках госзадания ФГУ ФНЦ НИИСИ РАН (Фундаментальные исследования 47 ГП) по теме № FNEF-2021-0001 (0580-2021-0001), рег. № 121031300047-6.

Литература

1. Embedded Systems in the Internet of Things. URL: https://www.benisontech.com/embedded-in-the-internet-of-things-2/ (дата обращения: 25.05.2021).

2. Hobbs C. Embedded Software Development for Safety-Critical Systems. 2019, 364 p. DOI: 10.1201/b18965.

3. Bertolotti I., Manduchi G. Real-Time Embedded Systems: Open-Source Operating Systems Perspective. USA, Florida, Boca Raton, CRC Press publ., 2012, 534 p. DOI: 10.1201/b11651.

4. Годунов А.Н. Операционная система реального времени Багет 3.0 // Программные продукты и системы. 2010. № 4. С. 15–19.

5. Годунов А.Н., Солдатов В.А. Операционные системы семейства Багет (сходство, отличия и перспективы) // Программирование. 2014. № 5. С. 68–76.

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

7. Drozdov A.Yu., Fonin Yu.N., Perov M.N., Vedishcheva T.S., Novoselova Yu.K. An approach to cross-platform drivers development. Proc. Intern. Conf. EnT, 2015, pp. 54–57. DOI: 10.1109/ENT.2015.14.

8. Lefebvre Y. An embedded platform-agnostic solution to deploy graphical applications. SAE Technical Paper Series, 2011, no. 2011-01-2551. DOI: 10.4271/2011-01-2551.

9. Palatty J.J., Edireswarapu S.P.C., Sivraj P. Performance analysis of FreeRTOS based video capture system. Proc. III ICECA, 2019, pp. 595–599. DOI: 10.1109/ICECA.2019.8822071.

10. 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. DOI: 10.1109/WCICA.2018.8630561.

11. Barry P., Crowley P. Modern Embedded Computing Designing Connected, Pervasive, Media-Rich Systems. Morgan Kaufmann publ., 2012, 544 p.

12. Богданов А.Ю. Опыт применения платформы прототипирования на ПЛИС «Protium» для верификации микропроцессоров // Тр. НИИСИ РАН. 2017. Т. 7. № 2. С. 46–49.

13. Zhadchenko A.V., Mamrosenko K.A., Giatsintov A.M. Porting X windows system to operating system compliant with portable operating system interface. IJACSA, 2020, vol. 11, no. 7, pp. 17–22. DOI: 10.14569/IJACSA.2020.0110703.

Software & Systems
DOI: 10.15827/0236-235X.135.433-439

Received 17.06.21
2021, vol. 34, no. 3, pp. 433–439

Approaches to providing data visualization on devices using modern real time operating systems

P.S. Bazhenov [1], *Leading Programmer, bps@niisi.ras.ru*
A.M. Giatsintov [1], *Ph.D. (Engineering), Chief Researcher, giatsintov@niisi.ras.ru*
K.A. Mamrosenko [1], *Ph.D. (Engineering), Head of the Research Center, mamrosenko_k@niisi.ras.ru*

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

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 method also takes into account the features related to using hardware interrupts and estimating the frequency 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 approaches 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 simulation speed.

The proposed method was tested during the development of the display controller driver for the home made RTOS, specifically – an X Window System display driver.

Keywords: visualization, real-time operating systems, driver, display controller, X Window System, embedded systems, POSIX, ARINC.

Acknowledgements. *Publication is made as part of national assignment for SRISA RAS (fundamental scientific research 47 GP) on the topic no. FNEF-2021-0001 (0580-2021-0001), reg. no. 121031300047-6.*

References

1. *Embedded Systems in the Internet of Things*. Available at: https://www.benisontech.com/embedded-in-the-internet-of-things-2/ (accessed May 25, 2021).

2. Hobbs C. *Embedded Software Development for Safety-Critical Systems*. 2019, 364 p. DOI: 10.1201/b18965.

3. Bertolotti I., Manduchi G. *Real-Time Embedded Systems: Open-Source Operating Systems Perspective*. USA, Florida, Boca Raton, CRC Press publ., 2012, 534 p. DOI: 10.1201/b11651.

4. Godunov A.N. Real-time operating system Baget 3.0. *Software and Systems*, 2010, no. 4, pp. 15–19 (in Russ.).

5. Godunov A.N., Soldatov V.A. Operating systems of the baguette family (likeness, differences and perspectives). *Programming and Computer Software*, 2014, no. 5, pp. 68–76 (in Russ.).

6. Pugin K.V., Mamrosenko K.A., Giatsintov A.M. Visualization of graphic information in the general-purpose operating systems. *RENSIT*, 2019, no. 1, pp. 217–224. DOI: 10.17725/rensit.2019.11.217 (in Russ.).

7. Drozdov A.Yu., Fonin Yu.N., Perov M.N., Vedishcheva T.S., Novoselova Yu.K. An approach to cross-platform drivers development. *Proc. Intern. Conf. EnT*, 2015, pp. 54–57. DOI: 10.1109/ENT.2015.14.

8. Lefebvre Y. An embedded platform-agnostic solution to deploy graphical applications. *SAE Technical Paper Series*, 2011, no. 2011-01-2551. DOI: 10.4271/2011-01-2551.

9. Palatty J.J., Edireswarapu S.P.C., Sivraj P. Performance analysis of FreeRTOS based video capture system. *Proc. III ICECA*, 2019, pp. 595–599. DOI: 10.1109/ICECA.2019.8822071.

10. 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. DOI: 10.1109/WCICA.2018.8630561.

11. Barry P., Crowley P. *Modern Embedded Computing Designing Connected, Pervasive, Media-Rich Systems*. Morgan Kaufmann publ., 2012, 544 p.

12. Bogdanov A.Yu. Experience in applying the Protium SOC prototyping platform for verification of microprocessors. *Proc. SRISA RAS*, 2017, vol. 7, no. 2, pp. 46–49 (in Russ.).

13. Zhadchenko A.V., Mamrosenko K.A., Giatsintov A.M. Porting X windows system to operating system compliant with portable operating system interface. *IJACSA*, 2020, vol. 11, no. 7, pp. 17–22. DOI: 10.14569/IJACSA.2020.0110703.

Для цитирования

Баженов П.С., Гиацинтов А.М., Мамросенко К.А. Подходы к обеспечению визуализации данных на устройствах с использованием современных операционных систем реального времени // Программные продукты и системы. 2021. Т. 34. № 3. С. 433–439. DOI: 10.15827/0236-235X.135.433-439.

For citation

Bazhenov P.S., Giatsintov A.M., Mamrosenko K.A. Approaches to providing data visualization on devices using modern real time operating systems. *Software & Systems*, 2021, vol. 34, no. 3, pp. 433–439 (in Russ.). DOI: 10.15827/0236-235X.135.433-439.