УДК 004.4’22
DOI: 10.15827/0236-235X.128.595-600
Дата подачи статьи: 22.07.19
2019. Т. 32. № 4. С. 595–600

Подходы к разработке и отладке симуляторов на основе QEMU с помощью высокоуровневого языка описания архитектур PPDL

А.Ю. Дроздов [1], *д.т.н., профессор, alexander.y.drozdov@gmail.com*
Ю.Н. Фонин [1], *научный сотрудник, fonin.iun@mipt.ru*
М.Н. Перов [1], *лаборант, coder@frtk.ru*
А.С. Герасимов [1], *лаборант, samik.mechanic@gmail.com*

[1] *Московский физико-технический институт (технический университет), г. Долгопрудный, 141701, Россия*

В статье описывается подход к разработке и отладке симуляторов на базе бинарной трансляции QEMU (Quick EMUlator), основанный на использовании высокоуровневого языка описания архитектур PPDL (Processor and Periphery Description Language). Применение бинарной трансляции в симуляторах позволяет ускорить работу на несколько порядков относительно симуляторов-интерпретаторов инструкции, а использование именно QEMU предоставляет широкий спектр возможностей как для отладки ПО, так и для динамического анализа приложений. Поэтому симуляторы на основе бинарной трансляции, в частности QEMU, представляют интерес для разработчиков как систем на кристалле уровня системы, так и встроенного ПО.

Однако процесс разработки бинарных трансляторов более сложен и трудоемок, чем разработка симулятора-интерпретатора инструкций. В отличие от симулятора-интерпретатора для создания QEMU-симулятора инструкции моделируемого процессора необходимо описать в виде последовательности так называемых tcg-микроинструкций. Основная сложность заключается в отладке такого симулятора, поскольку последовательность tcg-инструкций не исполняется непосредственно, а транслируется в двоичный код хост-машины. Поэтому в отличие от интерпретатора невозможно использовать стандартные средства отладки для локализации ошибок, допускаемых программистом при описании инструкций в виде tcg-кода.

Упростить задачу разработки QEMU-симулятора можно с помощью языка описания архитектур PPDL. Компилятор PPDL автоматически генерирует как симулятор-интерпретатор, так и набор компонент для QEMU-симулятора. Симулятор-интерпретатор генерируется в виде исходного кода на C++, что дает возможность отладки PPDL-описания ядра с помощью стандартного отладчика C\C++, например gdb или Microsoft Visual Studio. После отладки с помощью интерпретатора PPDL-описания посредством компилятора PPDL можно сгенерировать tcg-описания инструкций для QEMU. Таким образом, использование PPDL позволяет избежать отладки симулятора на уровне tcg-кода и, как следствие, существенно ускорить и упростить процесс разработки.

Ключевые слова: *симулятор, QEMU, языки описания архитектур.*

Проектирование современных микропроцессорных систем на кристалле невозможно без создания и использования программных симуляторов. Симулятор используется как для верификации аппаратной платформы, так и для разработки, отладки и оптимизации приложений до момента получения микросхемы с фабрики [1]. В ряде случаев симулятор позволяет получить детальную информацию о работе программы, которую невозможно получить при запуске программы на микропроцессоре, например, информацию о длительности и причине блокировок конвейера для каждой исполняемой инструкции.

При отладке больших приложений принципиальным становится вопрос скорости работы симулятора. Одним из способов создания быстрого симулятора является применение бинарной трансляции – преобразований последовательности инструкций эмулируемой архитектуры в инструкции архитектуры процессора, на котором осуществляется эмуляция. Такой подход позволяет достичь скорости работы симулятора до нескольких сотен мегагерц. Существует ряд симуляторов с открытым кодом, среди которых самый известный – QEMU (quick emulator) [2–4], также известны реализации симуляторов на основе компилятора LLVM (low level virtual machine) [5].

Ключевым недостатком симуляторов на основе бинарной трансляции является сложность разработки и отладки симулятора, поскольку

код инструкций представляется не в виде C/C++ текста, а как последовательность микроинструкций, которые затем будут транслироваться в код целевой машины. При этом отладка оттранслированного кода стандартными отладочными инструментами, такими как, например, GNU Debugger [6], сокращенно GDB, невыполнима, так как исполняемый бинарный код невозможно связать с исходным кодом программы. Для отладки таких симуляторов по сути доступна только возможность трассировки через вывод лога на консоль или в файл. Однако даже вывод лога не всегда позволяет получить всю необходимую информацию для быстрой локализации ошибок.

Ускорить разработку симуляторов на основе бинарной трансляции можно посредством высокоуровневых языков описания архитектур [7]. Эти языки содержат в себе описание, двоичное представление, ассемблерную мнемонику и семантику инструкции, что позволяет реализовать автоматическую генерацию алгоритмов бинарной трансляции, а также сгенерировать структуры и функции для поддержки подсистем отладки. Одним из таких высокоуровневых языков является PPDL (processor and periphery description language) [8].

В статье описываются метод внедрения новой архитектуры в QEMU и подход к отладке созданного эмулятора, основанные на использовании языка описания архитектур высокого уровня PPDL.

Особенности архитектуры симулятора QEMU

Ключевое отличие симулятора на основе QEMU от обычного C++/SystemC-симулятора в том, что операции в QEMU должны представляться в виде микроопераций фронтенда Tiny Code Generator, сокращенно TCG [2]. Далее эти микрооперации будем называть TCG-операциями.

Представление большей части кода в виде TCG-операций – основная задача для интеграции архитектуры в симулятор, поскольку такое начальное преобразование кода обеспечивает значительный прирост производительности при эмуляции и работе процессора на QEMU. Это объясняется тем, что генератору TCG требуется меньше машинного времени и прочих технических затрат на преобразование кода в промежуточное представление.

Процесс преобразования кода наглядно представлен на схемах (рис. 1–3). На входе TCG-модуля имеется некий гостевой код, команды которого относятся к поддерживаемой в QEMU архитектуре.

Функция gen_intermediate_code() осуществляет преобразование гостевого кода в промежуточное представление QEMU, которое состоит из TCG-операций.

Из промежуточного кода функция tcg_gen_code() осуществляет перевод TCG-операций в код целевой платформы, на которой запущен симулятор.

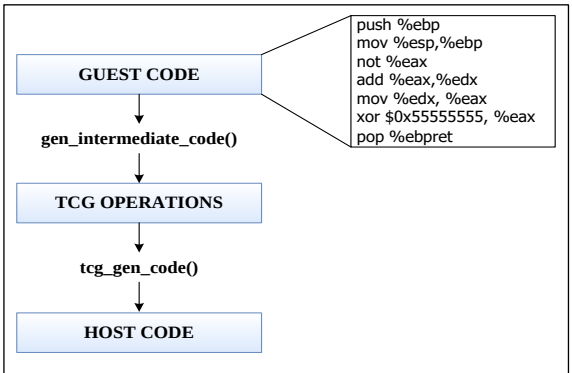

Рис. 1. Представление гостевого кода

Fig. 1. A guest code representation

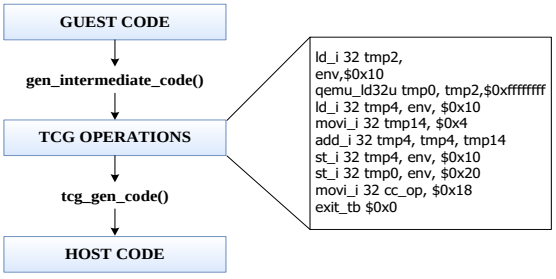

Рис. 2. Промежуточное представление модуля TCG

Fig. 2. TCG module intermediate view

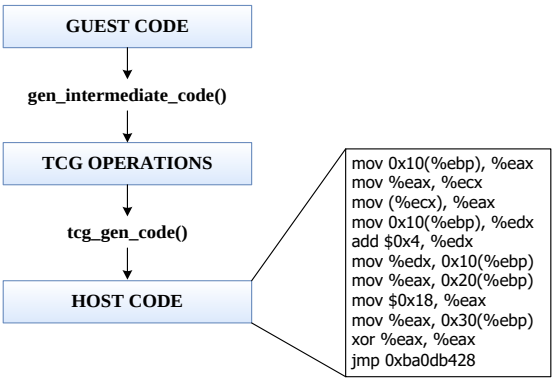

Рис. 3. Код целевой машины

Fig. 3. A target machine code

Описание всех доступных TCG-операций приведено в файле /tcg/README исходных кодов QEMU, а определение функций, реализующих эти операции, содержится в файле /tcg/tcg-op.h. Простейший вид TCG-функции: tcg_gen_<op-name>_tl (TCGv ret, TCGv arg1, TCGv arg2), где <op-name> – имя операции (например add); ret – имя TCG-переменной, в которую будет записан результат операции; arg1 и arg2 – операнды. Приставка _tl указывает модулю, что размер операндов генератор выбирает сам (TCGv_i32 для 32-битной архитектуры и 32-битных значений, TCGv_i64 для 64-битных значений).

Зная имена операций, не составляет труда определить нужную функцию генератора, осуществляющую соответствующее действие.

В некоторых случаях высокоуровневой операции нет однозначного соответствия в виде одной TCG-операции, тогда приходится составить соответствие в виде цепочки TCG-операций.

Возникают и более сложные ситуации, когда самостоятельное преобразование Си-представления какой-либо процессорной функции в промежуточное TCG-представление неэффективно. Для таких случаев существует возможность возложить работу по преобразованию функции на генератор TCG. Этот механизм называется хелпер-функциями.

Хелпер-функции (helper functions) – это Си-функции, которые могут быть вызваны напрямую из кода, представленного в виде TCG-операций. Механизм состоит в использовании специальных препроцессорных команд как указание на то, что TCG должен сам преобразовать данную функцию на языке Си в TCG-код. Компиляция хелпер-функций осуществляется на этапе компиляции, собственно, QEMU, а на этапе бинарной трансляции генерируется лишь код вызова данной функции.

Задача по встраиванию в QEMU-симулятор новой архитектуры состоит в реализации алгоритма декодирования инструкций и написания кода представления инструкции в виде последовательности TCG-операций. При этом инструментов, позволяющих выполнить, например, пошаговую трассировку исполнения кода, генерируемого из TCG-операций в режиме отладки, не существует. Есть возможность выводить трассы с помощью вставки функций-хелперов, однако этот метод достаточно трудоемкий и допускает возможность внесения ошибок в TCG-код симулятора. Поэтому процесс разработки симулятора на основе QEMU является более сложной и трудоемкой задачей, чем процесс разработки симулятора на языках C или C++, реализуемого в виде интерпретатора инструкций.

Разработка QEMU-симулятора с помощью PPDL

Процесс разработки QEMU-симулятора можно существенно упростить, если использовать языки описания архитектур высокого уровня. Одним из них является PPDL – язык для описания ядер микропроцессоров высокого уровня. В PPDL инструкции описываются в явном виде, причем описание инструкции включает в себя, помимо описания семантики, описания ассемблерной мнемоники и бинарного представления инструкции, включая описание полей аргументов инструкции. Благодаря такому представлению компилятор языка PPDL автоматически генерирует из описания архитектуры два симулятора, первый в виде C++ класса, а второй в виде набора компонент для QEMU-симулятора новой архитектуры, а также ассемблер, дизассемблер и набор структур и функций для отладчика. В PPDL для каждой инструкции описываются двоичное представление инструкции, ассемблерная мнемоника инструкции и семантика. В части описания семантики инструкций PPDL можно считать C-подобным языком. Кроме того, в PPDL есть возможность описывать индексируемые множества ресурсов или операций. Например, можно определить множество регистров или множество выражений для вычисления адреса при обращении к памяти. Для каждого множества, как и для инструкции, могут быть заданы ассемблерная мнемоника, двоичный код каждого элемента, а также семантика (в случае, если элемент множества – выражение) или имя ресурса (например, регистра или флага). Использование множеств в качестве типов аргументов инструкций позволяет существенно упростить описание архитектуры за счет переиспользования одних и тех же групп аргументов в различных инструкциях.

Наличие в PPDL описания, ассемблерной мнемоники, а также двоичного представления и семантики полезно не только для генерации системного ПО, но и для формирования трасс исполнения инструкций в симуляторах.

Для обоих автоматически генерируемых PPDL-компилятором симуляторов (C++ и QEMU) существует универсальный механизм генерации трасс выполнения инструкций. Трасса инструкций позволяет после каждой ис-

полненной инструкции вывести следующую информацию:

– адрес инструкции;
– бинарный код инструкции;
– дизассемблер инструкции;
– новые значения регистров, если после выполнения инструкций были изменены значения регистров процессора.

В случае, если симуляторы C++ и QEMU полностью эквивалентны, трассы одной и той же программы, исполненной на обоих симуляторах, будут одинаковыми. В случае расхождения работы симуляторов можно достаточно легко определить конкретную инструкцию посредством сравнения двух трасс.

Возможность автоматической генерации двух симуляторов позволяет выполнить отладку PPDL-описания архитектуры с помощью симулятора C++, который, в отличие от QEMU-симулятора, позволяет использовать стандартные инструменты отладки, например GDB, или отладчик Visual Studio. После отладки с помощью симулятора C++ PPDL-описания архитектуры генерируется код симулятора QEMU, а затем осуществляется его тестирование. Если поведение QEMU-симулятора отличается от поведения симулятора C++, то для локализации ошибки сравниваются их трассы. Сравнение трасс позволяет однозначно определить первую инструкцию, на которой различается поведение, и начать детальное изучение поведения именно этой инструкции. Поведение инструкции в QEMU можно исследовать с помощью вставки отладочных функций-хелперов.

Разработка QEMU-симулятора процессора NM6407

Данный подход был применен при разработке симулятора микропроцессора NM6407 семейства NeuroMatrix компании НТЦ «Модуль». Процессор специального назначения NM6407 [9, 10] предназначен для высокоскоростной обработки сигналов и содержит инструкции векторных операций как с фиксированной, так и с плавающей точкой. Принципиальной с точки зрения разработки симулятора особенностью NM6407 является наличие аппаратных очередей FIFO (First in, First out) для выполнения быстрых операций матрично-векторных умножений. Такие очереди не представлены в списке базовых элементов процессоров в QEMU, и для работы с ними нет предусмотренных TCG-операций. Очередь решено было представить в виде структуры следующего вида:

```
typedef struct FIFO_64_9 {
        int32_t mReadCnt;
        int32_t mWriteCnt;
        int32_t setReadCnt;
        int32_t setWriteCnt;
        uint64_t fifo[9];
} FIFO_64_9;
```

где 64 – размер элементов FIFO; 9 – размер очереди; mReadCnt и mWriteCnt – маркеры для чтения/записи в очередь.

В языке PPDL присутствует такая абстракция, как FIFO-блок, что существенно упростило процесс описания архитектуры NM6407 на PPDL. В симуляторе-интерпретаторе FIFO-блок реализован с помощью класса-шаблона, в котором перегружены операторы присваивания и возвращения значения (оператор «()» в C++). Это существенно упростило генерацию кода симулятора-интерпретатора, так как работа с FIFO с точки зрения кода C++ практически не отличалась от работы с регистрами или флагами.

Симулятор QEMU не поддерживает работу с классами C++. Поскольку представление данных процессора в QEMU ограничено использованием переменных типа TCGv (простая структура, содержащая лишь число типа int) и переменных типа TCGv_ptr (переменная, содержащая указатель на некоторую область памяти эмулятора), решением о представлении очередей FIFO будет использование последней в связи со сложностью структуры очереди.

При этом возникает дополнительная сложность – генерация функций для работы с FIFO, так как использование указателей вызывает необходимость выполнения многочисленных операций работы с указателями. В силу этого данную работу было решено переложить на генератор TCG – использовать хелпер-функции.

Одним из нюансов также является отсутствие полиморфизма в языке Си, поэтому для очередей различных размеров невозможна реализация перегрузок функций. Для разрешения конфликта были использованы препроцессорные вставки и склейки, доступные для хелпер-функций.

Заключение

Опыт разработки и последующего использования симулятора NM6407 на основе QEMU с помощью языка PPDL показал как интерес к симуляторам на основе бинарной трансляции со стороны разработчиков аппаратных платформ, так и реализуемость подхода к разработке QEMU-симуляторов с использованием высокоуровневых языков описания архитек-

тур. Использование PPDL также существенно упростило дальнейшую поддержку симулятора, в частности, модификации симулятора, обусловленные необходимостью повторить недокументированные нюансы работы реальной микросхемы, выпущенной в «кремнии».

Дальнейшая работа будет направлена на исследование подходов к разработке и отладке многоядерных гетерогенных архитектур на основе симулятора QEMU. Базовая версия QEMU в настоящее время не поддерживает возможность создания симуляторов гетерогенных архитектур, однако имеется открытое решение QEMU-TLM [11], которое является интеграцией QEMU, SystemC и технологии TLM (Transaction Level Modelling) и позволяет создавать симуляторы многопроцессорных систем. Кроме того, применение SystemC дает возможность моделировать не только ядра процессоров, но и модели периферийных устройств, сложных подсистем памяти, включая кэш-память [12], и аппаратных ускорителей.

Литература

1. Речистов Г.С., Юлюгин Е.А., Иванов А.А., Шишпор П.Л., Щелкунов Н.Н., Гаврилов Д.А. Программное моделирование вычислительных систем. 2016. 401 с. URL: http://atakua.doesntexist.org/wordpress/simulation-course-russian/ (дата обращения: 03.07.2019).

2. QEMU Emulator User Documentation URL: https://qemu.weilnetz.de/doc/qemu-doc.html (дата обращения: 03.07.2019).

3. Bellard F. QEMU, a fast and portable dynamic translator. Proc. Conf. USENIX, ATEC, Anaheim, USA, 2005, pp. 41–46.

4. Shen S.T., Lee S.Y., Chen C.H. Full system simulation with QEMU: an approach to multi-view 3D GPU design. Proc. ISCAS, IEEE, Paris, France, 2010, pp. 3877–3880. DOI: 10.1109/ISCAS.2010.5537690.

5. Cook S. AAPSim: Implementing a LLVM Based Simulator. Embecosm, 2016. URL: http://llvm.org/devmtg/2016-01/slides/fosdem16-aapsim.pdf (дата обращения: 03.07.2019).

6. GNU Debugger description. URL: https://www.gnu.org/software/gdb/ (дата обращения: 03.07.2019).

7. Рубанов В.В. Обзор методов описания встраиваемой аппаратуры и построения инструментария кросс-разработки // Тр. ИСП РАН. 2008. Т. 15. С. 7–40.

8. Фонин Ю.Н. Применение языка PPDL для автоматической генерации средств разработки процессоров // Информационные технологии. 2017. №. 8. Т. 23. С. 583–588.

9. Бирюков А.А., Таранин М.В., Таранин С.В. Процессор 1879ВМ6Я. Реализация глубоких сверточных нейронных сетей // DSPA: Вопросы применения цифровой обработки сигналов. 2017. № 4. Т. 8. С. 191–195.

10. Мушкаев С.В., Бродяженко А.В., Болотников А.А. Вычислительные ресурсы процессоров NeuroMatrix с плавающей точкой в задачах обработки больших потоков данных // Наноиндустрия. 2018. № 9. С. 110–118. DOI: 10.22184/1993-8578.2018.82.110.118.

11. Yeh T.C., Lin Z.Y., Chiang M.C. Enabling TLM-2.0 interface on QEMU and SystemC-based virtual platform. Proc. ICICDT, Kaohsiung, Taiwan, 2011. DOI: 10.1109/ICICDT.2011.5783207.

12. Cheung E., Hsieh H., Balarin F. Memory Subsystem Simulation in Software TLM/T Models. Proc. 14th ASP-DAC, Yokohama, Japan, 2009, pp. 811–816.

Software & Systems Received 22.07.19
DOI: 10.15827/0236-235X.128.595-600 2019, vol. 32, no. 4, pp. 595–600

Approaches to the development and debugging QEMU simulators using the high-level architecture describing language PPDL

A.Yu. Drozdov [1], *Dr.Sc. (Engineering), Professor, alexander.y.drozdov@gmail.com*
Yu.N. Fonin [1], *Research Associate, fonin.iun@mipt.ru*
M.N. Perov [1], *Laboratory Assistant, coder@frtk.ru*
A.S. Gerasimov [1], *Laboratory Assistant, samik.mechanic@gmail.com*

[1] *Moscow Institute of Physics and Technology, Dolgoprudniy, 141700, Russian Federation*

Abstract. The paper describes an approach to the development and debugging simulators based on QEMU (Quick EMUlator) binary translation. This approach is based on using PPDL (Processor and Periphery Description Language) that is a high-level architecture describing language. Simulations based on binary translation work several times faster in contrast to instruction interpreters while providing a wide range of possibilities

for software debugging, as well as for dynamic analysis of applications. Thus, binary translation simulators based on QEMU in particular are of high interest either to system-level SoC (System on Crystal) developers and to embedded software developers.

However, developing of binary translators is a more complicated and more time-consuming task compared to instruction interpreter development. Development of the QEMU simulator assumes the implementation of instructions of the simulated processor as a sequence of so-called tcg micro-operations. Tcg micro-operations are not executed directly, rather used for binary translation to the instructions of the host machine. Therefore, there is no possibility to debug tcg description of instructions using standard debuggers.

It is possible to simplify QEMU simulator developing using PPDL language. PPDL compiler generates two simulators from PPDL description of processor: an interpreter and a QEMU component kit. The compiler generates an interpreter as a C++ source code. With generated C++ code, any debugger like gdb or Microsoft Visual Studio can debug PPDL description. Than from the same description PPDL compiler generates the QEMU description of a processor representing instructions as a sequences of tcg micro-operation. Due to PPDL, developers can avoid debugging of the tcg processor description and therefore accelerate development of a QEMU based simulator.

Keywords: simulator, QEMU, architecture describing languages.

References

1. Rechistov G.S., Yulugin E.A., Ivanov A.A., Shishpor P.L., Schelkunov N.N., Gavrilov D.A. *Software Modeling of Computer Systems*. 2016, 401 p. Available at: http://atakua.doesntexist.org/wordpress/simulation-course-russian/ (accessed July 03, 2019).

2. *QEMU Emulator User Documentation.* Available at: https://qemu.weilnetz.de/doc/qemu-doc.html (accessed July 03, 2019).

3. Bellard F. QEMU, a fast and portable dynamic translator. *Proc. USENIX Annual Technical Conf. (ATEC)*. Anaheim, USA, 2005, pp. 41–46.

4. Shen S.T., Lee S.Y., Chen C.H. Full system simulation with QEMU: an approach to multi-view 3D GPU design. *Proc. ISCAS, IEEE*. Paris, France, 2010, pp. 3877–3880. DOI: 10.1109/ISCAS.2010.5537690.

5. Cook S. *AAPSim: Implementing a LLVM Based Simulator*. Embecosm, 2016. Available at: http://llvm.org/devmtg/2016-01/slides/fosdem16-aapsim.pdf (accessed July 03, 2019).

6. *GNU Debugger Description*. Available at: https://www.gnu.org/software/gdb/ (accessed July 03, 2019).

7. Rubanov V.V. A review of methods for describing embedded hardware and building cross-development tools. *Proc. ISP RAS*. 2008, vol. 15, p. 7–40 (in Russ.).

8. Fonin Yu.N. The use of PPDL language for automatic generation of processor development tools. *Information Technologies*. Moscow, 2017, vol. 23, no. 8, pp. 583–588 (in Russ.).

9. Biryukov A.A., Taranin M.V., Taranin S.V. The processor is 1879VM6YA. Implementation of deep convolutional neural networks. *DSPA: Issues of Using Digital Signal Processing*. Moscow, 2017, no. 4, vol. 8, pp. 191–195 (in Russ.).

10. Mushkaev S.V., Brodyazhenko A.V., Bolotnikov A.A. Computational resources of NeuroMatrix processors with a floating point in the tasks of processing large data streams. *Nanoindustry*. Moscow, Tekhnosphera Publ., 2018, iss. 9, pp. 110–118. DOI: 10.22184/1993-8578.2018.82.110.118 (in Russ.).

11. Yeh T.C., Lin Z.Y., Chiang M.C. Enabling TLM-2.0 interface on QEMU and SystemC-based virtual platform. *IEEE Intern. Conf. on IC Design & Technology*. Kaohsiung, Taiwan, 2011. DOI: 10.1109/ICICDT.2011.5783207.

12. Cheung E., Hsieh H., Balarin F. Memory Subsystem Simulation in Software TLM/T Models. *Proc. 14th Asia South Pacific Design Automation Conf*. Yokohama, Japan, 2009, pp. 811–816.

Для цитирования

Дроздов А.Ю., Фонин Ю.Н., Перов М.Н., Герасимов А.С. Подходы к разработке и отладке симуляторов на основе QEMU с помощью высокоуровневого языка описания архитектур PPDL // Программные продукты и системы. 2019. Т. 32. № 4. С. 595–600. DOI: 10.15827/0236-235X.128.595-600.

For citation

Drozdov A.Yu., Fonin Yu.N., Perov M.N., Gerasimov A.S. Approaches to the development and debugging QEMU simulators using the high-level architecture describing language PPDL. *Software & Systems*. 2019, vol. 32, no. 4, pp. 595–600 (in Russ.). DOI: 10.15827/0236-235X.128.595-600.