УДК. 004.94 DOI: 10.22184/NanoRus.2019.12.89.350.352

# РАЗРАБОТКА ПРОГРАММНОЙ ПЛАТФОРМЫ KOPYCAT ЭМУЛЯЦИИ СЛОЖНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

# DEVELOPING SOFTWARE KOPYCAT PLATFORM FOR EMULATION OF COMPLEX COMPUTING SYSTEMS

**АРИСТОВ РОМАН СЕРГЕЕВИЧ**
***Руководитель группы***
*r.aristov@inforion.ru*

**ГЛАДКИХ АЛЕКСЕЙ АЛЕКСЕЕВИЧ**
***К. т. н. руководитель отдела***
*a.gladkih@inforion.ru*

**ДАВЫДОВ ВЛАДИМИР НИКИТИЧ**
***Инженер 1-й категории***
*v.davydov@inforion.ru*

**КОМАХИН МИХАИЛ ОЛЕГОВИЧ**
***Инженер аппаратного обеспечения***
*m.komakhin@inforion.ru*

*ООО «ИНФОРИОН»*
*https://www.inforion.ru*

**ARISTOV ROMAN S.**
*r.aristov@inforion.ru*

**GLADKIKH ALEXEI A.**
*a.gladkih@inforion.ru*

**DAVYDOV VLADIMIR N.**
*v.davydov@inforion.ru*

**KOMAKHIN MICHAEL O.**
*m.komakhin@inforion.ru*

*"INFORION" Ltd*
*https://www.inforion.ru*

Рассмотрена разработка программного обеспечения Kopycat, позволяющего проводить эмуляцию различных сложных систем. Приведены общие сведения по разработанному эмулятору. Кроме этого, показано сравнение разработанного программного обеспечения с существующими в настоящее время аналогами. Представлены преимущества и недостатки данного программного решения по сравнению с аналогами.
*Ключевые слова: СБИС; эмуляция; Kopycat; Qemu; GDB; Verilog.*
The paper considers the development of the "Kopycat" program emulation software and gives general information about the emulator. In addition, a comparison of the developed software with existing analogues has been presented as well as advantages and disadvantages of this software solution in comparison with analogues.
*Keywords: VLSI; emulation; Kopycat; Qemu; GDB; Verilog.*

При разработке электроники зачастую возникает потребность в тестировании изделия во время его проектирования. Отладка программной части на самом устройстве имеет ограничения. Так, например, целевая аппаратная платформа не всегда доступна в процессе проектирования устройства, более того, может потребоваться изменение аппаратной части. Этот процесс является долгим и дорогостоящим. В связи с этим возникает необходимость проводить тестирование программного обеспечения, а также встроенного программного обеспечения (далее — ПО, ВПО) в отсутствие аппаратной части. Тестирование ПО на компьютере разработчика невозможно из-за различий в процессорной архитектуре целевого устройства и персонального компьютера (далее — ПК), особенностей операционной системы, необходимости взаимодействия ПО с периферийными устройствами, которые не могут быть подключены к ПК разработчика. Для решения вышеописанных задач используются эмуляторы. Основными продуктами в этой сфере являются Qemu, Unicorn, Proteus, Keil emulator.

В то же время у инженеров — разработчиков микроэлектроники возникает задача синтеза списка соединений логических вентилей из поведенческой и структурной модели проектируемого устройства. В настоящее время для решения этих задач используются языки описания аппаратуры, такие как VHDL, Verilog и т. д. Кроме того, существует проект SystemC, позволяющий описывать модели на языке C++ [1]. Таким образом, для реализации полного цикла разработки требуется набор из нескольких САПР, реализующих аппаратную эмуляцию и высокоуровневый синтез. Каждый из вышеперечисленных продуктов занимает свою нишу.

Специалистами ООО «ИНФОРИОН» была разработана программная платформа для создания эмуляторов Kopycat. Данный фреймворк был реализован на языке программирования Kotlin (Java). Он позволяет осуществлять низкоуровневую программную эмуляцию произвольных аппаратных систем и их отладку через интерфейс RSP GDB [2]. Этот интерфейс встроен в большинство современных сред проектирования программного и внутреннего программного обеспечения. Он позволяет проводить отладку устройства через удаленный сетевой TCP/IP-порт. Таким образом, к эмулятору можно удаленно подключиться как непосредственно самим отладчиком GDB, который является в настоящее время стандартом де-факто, так и такими средами разработки и отладки, как Eclipse CDT,

System Workbench, IDA Pro, NetBeans, CodeLite и т. п.

Следует отметить и тот факт, что особенностью разработанного продукта является то, что моделирование происходит не на уровне электрических сигналов, а на уровне логических соединений шин. Достоинством такого подхода является существенно более высокое быстродействие по сравнению со средами моделирования электрических схем, позволяющих эмулировать микроконтроллеры, например ISIS Proteus. В сравнении с САПР ISIS Proteus Корусat использует другой способ моделирования, при котором состояние каждого из компонентов эмулятора определяется внешними событиями и не требует обновления состояния каждый такт работы эмулятора. Использование такой модели позволяет снизить нагрузку на вычислительные мощности ПК и получить превосходство скорости моделирования относительно ISIS Proteus. Кроме этого, такая событийно-ориентированная модель сокращает время разработки эмуляторов целевых аппаратных платформ для конечного пользователя. Однако в силу использования событийно-ориентированной модели в Корусat он имеет некоторые ограничения: моделирование электрических сигналов и использование SPICE-моделей устройств затруднено и неэффективно. Для решения подобных задач предпочтительнее использовать САПР ISIS Proteus.

```
{
  "top": true,
  "plugin": "AMDElanSC520", // Plugin name should be the same as file name (or full path from library start)
  "library": "amd",         // Directory where plugin places
  "params": [ ],            // Plugin parameters (constructor parameters if jar-plugin version)
  "ports": [ ],             // Plugin outer ports
  "buses": [                // Plugin internal buses
    { "name": "mem", "width": 32 },
    { "name": "io", "width": 16 }
  ],

  "modules": [    // Plugin internal components
    {
      "name": "x86",
      "plugin": "x86Core",
      "library": "cores",
      "params": {
        "config": "extras/amd/binaries/core.json"
      }
    },
    {
      "name": "uart",
      "plugin": "Uart",
      "library": "amd"
    }
  ],
  "connections": [  // Plugin connection between components
    [ "x86.ports.io", "uart.buses.io", "0x1000" ]
  ]
}
```

*Рис. 1. Пример структурного описания модуля в формате JSON*

```
class Uart(parent: Module, name: String) : Module(parent, name) {
    companion object { private val log = logger(ALL) }

    inner class Ports : ModulePorts( module: this) {
        val io = Port( name: "mem")
    }

    override val ports = Ports()

    val uartTx = File("temp/uart_tx.txt").outputStream()

    val UART_IO = object : Register(ports.io,  address: 0x0FAC, BYTE,  name: "UART_IO") {
        override fun read(ea: Long, ss: Int, size: Int): Long = 0
        override fun write(ea: Long, ss: Int, size: Int, value: Long) { uartTx.write(value.toInt()) }
    }

    val UART_STATUS = Register(ports.io,  address: 0x0FAD, BYTE,  name: "UART_STATUS", default = 0)
}
```

*Рис. 2. Пример поведенческого описания модуля на языке программирования Kotlin*

С другой стороны, для тестирования достаточно логической модели устройства в этой области используются эмуляторы Qemu и Unicorn. Они, так же как и Корусat, используют событийно-ориентированную модель и показывают высокие показатели быстродействия. Помимо этого, Qemu имеет большое сообщество разработчиков и, как следствие, хорошую поддержку. Несмотря на это, разработка периферийных модулей требует высокой подготовки специалиста и больших временных затрат. Одним из конкурентных преимуществ Корусat является наличие удобного SDK-разработчика и документации для создания новых вычислительных ядер и периферийных устройств [3, 4]. Корусat имеет API для интеграции со сторонними продуктами.

При этом возможно два подхода для создания новых модулей: первый — это структурный подход, второй — поведенческое описание. При структурном описании в файле в формате JSON указываются установленные модули и соединения между ними. Также структурное описание допустимо проводить и с помощью языка программирования Kotlin/Java. Такое описание может быть использовано, когда новый модуль собирается только из уже готовых (ранее разработанных), при этом не добавляется нового поведения или логической обработки в модуль. При поведенческом же описании в модуль могут быть добавлены обработчики записи или чтения с определенных адресов (к которому будет подключен модуль). Внутри обработчика при этом задается, каким образом модуль должен отрабатывать данное событие. Поведенческое описание может быть задано только на языке программирования Kotlin/Java.

*Таблица 1. Сравнение инструментов эмуляции и моделирования электрических схем*

| | Моделирование электрических сигналов | Синтез описания на Verilog | Интерфейс GDB | Быстродействие* | Сложность реализации новых модулей* |
|---|---|---|---|---|---|
| Qemu | Нет | Нет | Есть | высокое | высокая |
| ISIS Proteus | Есть | Нет | Нет | низкое | высокая |
| **Корусat** | **Нет** | **Планируется** | **Есть** | **среднее** | **средняя** |
| Keil emulator | Нет | Нет | Есть | крайне низкое | отсутствует |
| SystemC | Есть | Есть | Нет | низкое | высокая |

* Значения качественных характеристик приведены относительно элементов этой таблицы.

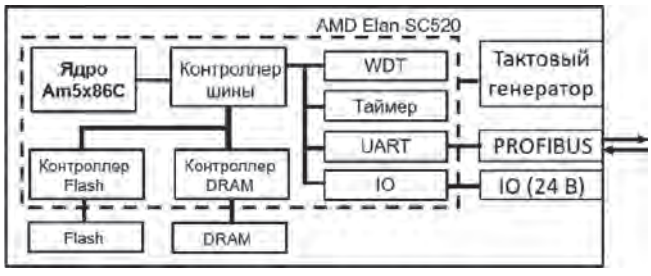


*Рис. 3. Архитектура устройства для эмуляции на базе микроконтроллера AMD Elan SC520 с архитектурой x86*

Пример структурного описания приведен на рис. 1, а на рис. 2 приведен пример поведенческого описания модуля.

Некоторые среды разработки, например Keil uVision, имеют встроенные средства эмуляции аппаратных платформ. Они имеют слабую поддержку и практически неприменимы для большинства задач, встающих перед разработчиком встроенного программного обеспечения. Это связано с тем, что в такого рода эмуляторах присутствует только возможность эмулировать ядро микроконтроллера, при этом периферийные модули, даже расположенные внутри микроконтроллера, никак не моделируются. Однако любая вычислительная система очень сильно связана со взаимодействием с периферийными модулями: передача данных по различным каналам связи (UART, SPI, One-wire, Ethernet, Profibus и т. п.), а также с выводом данных пользователю. Из приведенного выше видно, что такой эмулятор не представляет хоть сколько-нибудь значимого практического применения. Рассмотрим сравнение различных средств эмуляции и моделирования, которое приведено в сводной табл. 1.

Стоит отметить, что архитектура Korycat предусматривает синтез описания аппаратной платформы на языке Verilog. Например, в соответствии с разработанной моделью была спроектирована аппаратная платформа на базе микроконтроллера AMD Elan SC520 с архитектурой x86. Схема разработанной модели устройства для эмуляции приведена на рис. 3.

Кроме указанных преимуществ, эмулятор Korycat успешно использовался для проведения автоматизированного тестирования ВПО различных заказных модулей. Аппаратное обеспечение данных модулей было построено на базе процессоров с архитектурой ARM, x86, MIPS, V850ES, C166. Причем передача данных осуществлялась по таким интерфейсам, как Ethernet, UART, SPI, Profibus. С этой целью в эмуляторе были реализованы периферийные модули контроллеров этих интерфейсов. Также для проведения таких работ были реализованы и другие периферийные модули, необходимые для работоспособности системы. Среди них можно выделить программируемые контроллеры прерываний (PIC, Programmable Interrupt Controller), ATA-контроллер, NAND-контроллеры, SD-контроллеры, различные GPIO-контроллеры, таймеры различной сложности и функциональности. В общей сложности на данный момент для эмулятора разработано порядка 200 периферийных модулей для специальных заказных микроконтроллеров. При этом ведутся работы по реализации полнофункциональных модулей для эмуляции широко распространенных на текущий момент времени контроллеров от фирмы STMicroelectronics ST32F407 и др.

Таким образом, Korycat совмещает в себе преимущества быстрого создания новых модулей, быстрого моделирования целевой платформы, интеграции с другими инструментами разработки, отладки и тестирования. На данный момент компанией «ИНФОРИОН» реализовано несколько успешных проектов по автоматизированному тестированию и проектированию ВПО сложных вычислительных систем.

Авторы выражают благодарность сотрудникам ФГУ ФНЦ НИИСИ РАН Глушко А. А. и Яшину Г. А. за критические замечания и помощь в подготовке материалов данной статьи.

## ЛИТЕРАТУРА

1. Амирханов А. В., Гладких А. А., Глушко А. А., Михальцов Е. П., Родионов И. А., Столяров А. А. Особенности метода проектирования СБИС с учетом результатов моделирования технологического процесса // Труды научно-исследовательского института системных исследований российской академии наук — Москва: Федеральный научный центр «Научно-исследовательский институт системных исследований Российской академии наук», 2013. — С. 10–19.
2. Appendix E GDB Remote Serial Protocol. Электронный ресурс, режим доступа: https://sourceware.org/gdb/onlinedocs/gdb/Remote-Protocol.html.
3. Комахин М. О. Особенности реализации взаимодействия ядра с периферией в эмуляторах вычислительных систем // Наукоемкие технологии и интеллектуальные системы — Москва: МГТУ им. Н. Э. Баумана, 2018. — С. 246–250.
4. Валитов Р. Ш. Реализация программного эмулятора RISC процессорного ядра на примере V850ES // Наукоемкие технологии и интеллектуальные системы — Москва: МГТУ им. Н. Э. Баумана, 2018. — С. 121–126.
5. Давыдов В. Н. Разработка методик повышения надежности и безопасности встраиваемых вычислительных систем на базе комплексной низкоуровневой программной эмуляции // Наукоемкие технологии и интеллектуальные системы — Москва: МГТУ им. Н. Э. Баумана, 2017. — С. 56–62.