МИНОБРНАУКИ РОССИИ
Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет «Московский институт электронной техники»
Институт системной и программной инженерии и информационных технологий (СПИНТех)

Уманский Александр Александрович

Магистерская диссертация
по направлению 09.04.04 «Программная инженерия»

Создание методики и алгоритма генерации виртуального аппаратного обеспечения по спецификации

Студент _____
Уманский А.А.
Руководитель,
канд. техн. наук, доц. _____
Кононова А.И.

Москва, г. Зеленоград — 2021

Содержание

Список сокращений и условных обозначений

QOM	Qemu Object Model (Объектная модель Qemu)
ПМ	Программный модуль
ПО	Программное обеспечение
АО	Аппаратное обеспечение
ЯП	Язык программирования
GUI	Graphical User Interface
IDE	Интегрированная среда разработки
JSON	Формат описания структур данных в текстовом виде ключ → значение
PID	Уникальный идентификатор процесса в ОС
TDD	Test-driven development
HDL	Hardware description language
	Программный модуль анализа на недекларированные возможности

Словарь терминов

Кроссплатформенный: программа, которая может запускаться на различных операционных системах и/или архитектурах процессоров

Программная закладка: подпрограмма, либо фрагмент исходного кода, скрытно внедренный в исполняемый файл

Динамическая трасса: дерево вызванных программой функций во время конкретного ее исполнения

Статическая трасса: дерево функций программы, которые объявлены для вызова

Отладчик: программа, в контексте которой запускается другая программа для локализации и устранения ошибок в контролируемых условиях

Отладка: процесс локализации и устранения ошибок программы в контролируемых условиях

Удаленная отладка: процесс отладки программы, запущенной вне контекста отладчика

Препроцессор: программа-макропроцессор, обрабатывающая специальные директивы в исходном коде и запускающаяся до компилятора

Препроцессирование: процесс обработки исходного кода препроцессором

Открытое ПО: ПО с открытым исходным кодом, который доступен для просмотра, изучения и изменения

Сериализация: процесс перевода определеного типа данных программы в некоторый формат

Десериализация: процесс перевода данных, находящихся в некотором формате, во внутренний тип данных программы

Скрипт: программа, обычно на интерпретируемом языке программирования, выполняющая конкретное действие

Сигнатура функции: объявление функции, в которое входит имя функции, количество входных параметров и их тип

Сборка: процесс компиляции, линковки и публикации программного обеспечения из исходных кодов

Рефакторинг: процесс улучшения кода без введения новой функциональности. Результатом является чистый код с улучшенным дизайном

Релизная сборка: сборка программы происходит без отладочных символов,

обычно с использованием техник оптимизации кода

Терминал: то же, что и консоль

Антиотладка: набор методов детектирования отлаживаемой программы окружения отладки и препятствование ей

Мультитаскинг: возможность программы или операционной системы обеспечивать возможность параллелльного исполнения задач

Сверхвысокоуровневый ЯП: классификация языков программирования, к данной категории относятся языки программирования, позволяющие описать задачу не на уровне «как нужно сделать», а на уровне «что нужно сделать»

Source-to-source: Компиляция исходного кода некоторого языка в исходный код другого языка. Во время компиляции языка данным способом может происходить несколько итераций преобразования, пока последний язык в цепочке преобразований не будет скомпилирован в машинный код или интерпретирован

Актуальность исследования:

Компьютеризация и информатизация различных сфер общества и производства неизменно сопровождается созданием специфических программных и аппаратных средств.

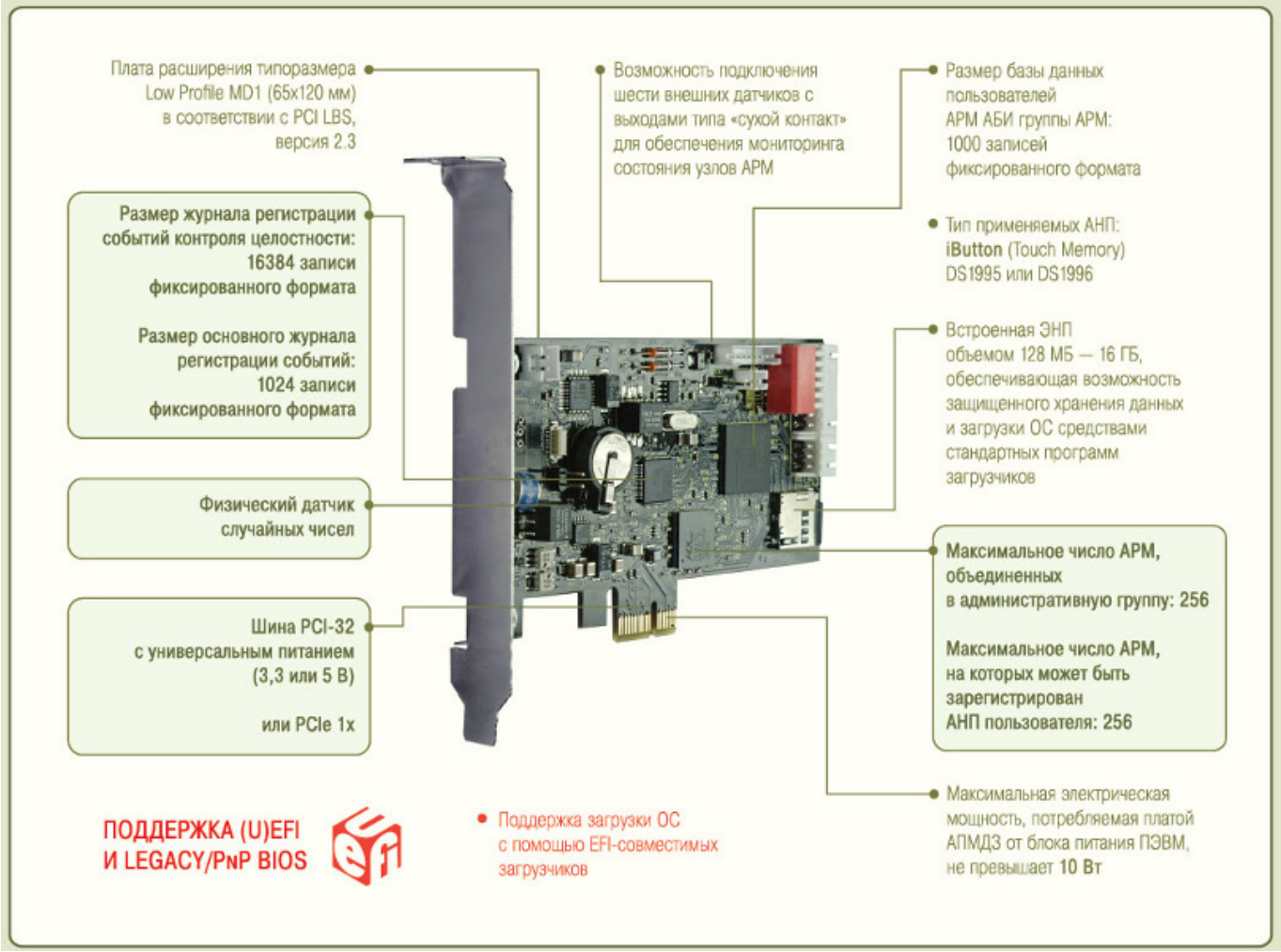

Рисунок 1 — Пример специфического аппаратного обеспечения: АПМДЗ Максим-М1

Разработка аппаратного обеспечения практически никогда не обходится без создания прикладного програмного обеспечения. Правильное проектирование архитектуры ПО и выделение уровней абстракции позволяет разрабатывать программное обеспечение и аппаратное обеспечение параллельно, но до определенного уровня.

В теории теория и практика не различаются, но не на практике. Для тестирования и отладки прикладного ПО в конечном счете понадобится аппаратное обеспечение, но зачастую им сложно обеспечить всю команду разработчиков, особенно на ранних стадиях. Причиной этому могут быть экономические, логистические, производственные или временные проблемы.

Но даже если обеспечить всех разработчиков необходимыми стендами, то при проявлении проблем во взаимодействии ПО и АО, аппаратное обеспечение придется отправлять на доработку, после чего снова печатать, доставлять и собирать стенды, что неизбежно приводит к экономическим и временным издержкам, которые, в свою очередь, повышают затраты на завершение проекта в срок.

Исправить и запустить ПО на поздних этапах проектирования легче, чем исправить и перевыпустить схему.

Причины сложившейся ситуации:

1) отсутствие формализованного подхода к созданию эмуляторов аппаратного обеспечения;
2) отсутствие универсального метода написания эмуляторов аппаратного обеспечения;

Необходимость создания методики и алгоритма генерации виртуального аппаратного обеспечения по спецификации:

Отсутствие быстрого способа создания эмуляторов специализированного аппаратного обеспечения, что отражается экономических аспектах конечного продукта.

Новизна исследования:

Технология эмуляции существует давно, но на данный момент не существует инструмента, который позволил бы быстро и просто создавать эмуляторы любых аппаратных устройств.

Значимость исследования состоит в создании универсального инструмента по созданию эмуляторов аппаратного обеспечения.

Раздел 1. Создание аппаратного обеспечения

В представлении обывателя компьютер (системный блок) состоит из материнской платы, процессора, оперативной памяти, возможно видеокарты. Давно прошли те времена, когда пользователям компьютеров приходилось отдельно покупать такие модули расширения, как, например, математический сопроцессор. С течением времени все больше ранее внешних модулей становится частью материнской платы или самого процессора.

Но производство аппаратного обеспечения не ограничивается потребительским рынком. Специфические аппаратные решения требуются отдельным отраслям или организациям.

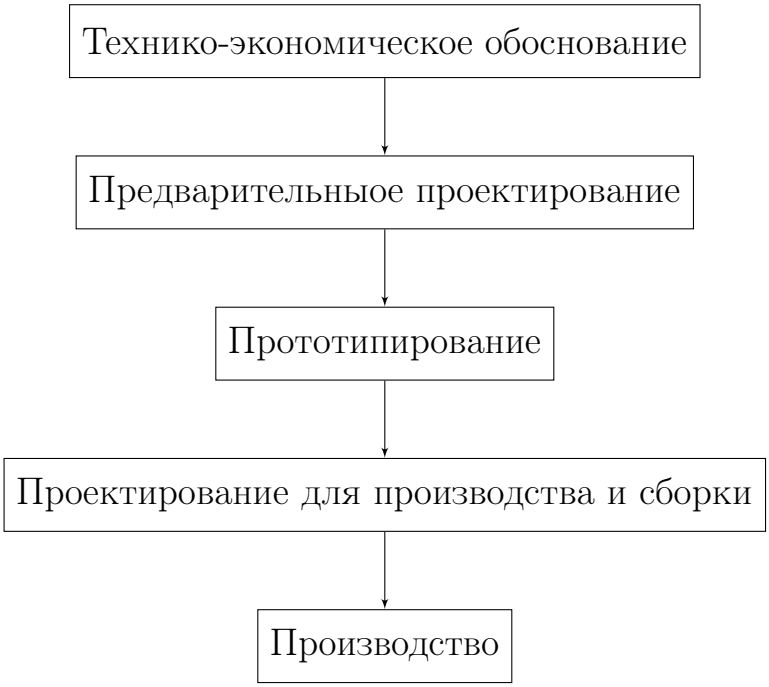

Рисунок 1.1 — Этапы создания аппаратного обеспечения

Создание аппаратного обеспечения трудоемкий процесс, непременно связанный с созданием прикладного программного обеспечения. В свою очередь, отсутствие возможности исполнения ПО с использованием АО серьезно усложняет отладку и тестирование конечного продукта.

1.1 Эмуляция аппаратного обеспечения

Тезис Чёрча-Тьюринга гласит: любая вычислимая (то есть та, которая может быть реализована на машине Тьюринга) функция вычислима машиной

Тьюринга. Физический тезис Чёрча-Тьюринга гласит: любая функция, которая может быть вычислена физическим устройством, может быть вычислена машиной Тьюринга.

Существуют различное отношение к тезису Чёрча-Тьюринга. Некоторые считают, что он может быть доказан, другие говорят, что он служит определением вычислений. Не смотря на то, что тезис до сих пор не доказан, его верность исходит из того, что любая, открытая на текущий момент реалистичная модель вычислений доказывала его правоту.

Тезис неразрывно связан с термином "эмуляция".

Эмуляция – это процесс имитирование поведения одного оборудования и/или программного обеспечения на другом оборудовании и/или программном обеспечении.

Эмулятор АО может использоваться как для имитирования совершенно другого оборудования, так и того, на котором она проводится. Например, большое количество принтеров умеет эмулировать линейку LaserJet компании Hewlett-Packard, так как большая часть ПО разработана как раз под LaserJet.

Эмуляторы аппаратного обеспечения бывают разного назначения:

Симуляторы логики Данный вид эмуляторов используется для исследования и верификации аппаратного обеспечения на различных уровнях:

- компонентном;
- логических вентилей;
- регистровых передач;

Высокоуровневые эмуляторы Высокоуровневая эмуляция – это набор методов эмулирования некоторых компонентов целой системы. Позволяет не исполнять инструкции или производить обработку данных один в один, как на целевом оборудовании, заменяя "горячие"пути обработки аналогичными по результату.

Одними из первых данный подход был использован в эмуляторах игровых консолей, где команды отрисовки трёхмерной сцены эмулировались не на процессоре, как все остальное оборудование консоли, а отдавались напрямую графическому процессору машины, на которой происходил процесс эмуляции.

Эмуляторы процессора Эмулируют конкретную архитектуру процессора, позволяя запускать программы, не предназначенные для физического компьютера. Зачастую умеют эмулировать не только процессор но и переферийные устройства.

Эмуляторы терминала Эмулируют терминал компьютера внутри некоторой архитектуры отображения данных на дисплее.

Сэмуляторы Являются симбиозом симуляторов и эмуляторов, который берет лучшее от двух миров. Аппаратное обеспечение описовается HDL языками, после чего данное описание оборудования симулируется на стендах. Первоначальная функциональная верификация производится через симуляцию на уровне регистровых передач или логических вентилей. В событийно-ориентированной симуляции инструкции последовательно исполняются процессором, потому что в превалирующем большинстве сценариев не возможно провести данную симуляцию параллельно. Последовательных подход приводит к долгим симуляциям, особенно в сложных системах на кристалле. После симуляции описание регистровых передач должно быть зашито в аппаратное обеспечение (FPGA, ASIC). Но идеализированное представление аппаратного обеспечения в симуляции отличается от реального аппаратного обеспечения. Отличие между симуляцией и работой аппаратного обеспечения является серьезной причиной применений эмуляции при проектировании. Преимуществом данного метода является:

- ускорение симуляции: часть сложной системы, не релевантная для симуляции переносится в эмулятор, что позволяет вынести из симуляции нерелевантные части;
- использование реального аппаратного обеспечения на ранних этапах проектирования;

1.2 Аппаратное обеспечение – физическое и виртуальное

Прикладная польза от эмуляции аппаратного обеспечения может быть достигнута не только при проектировании и разработке специализированного аппаратного обеспечения, но и, напрмер, для совместного использования одного физического устройства несколькими компьютерами (в основном виртуальными машинами). Представляя аппаратное обеспечение гостевой системе как физическим устройство, можно свести на нет затраты на написание специализированных

драйверов и утилизировать существующие, вынося обработку совместного использования в реализацию эмулируемого аппаратного обеспечения.

Например, можно запустить некоторое количество виртуальных машин на одной физической, и для каждой виртуальной машины использование графического ускорителя будет прозрачным, тогда как на самом деле управлением задачами отрисовки или обработки данных будет заниматься виртуальное устройство.

Также вынесение интерфейса реального устройства в виртуальное позволяет применить общение с программой-посредником, которая будет отдавать команды реальному устройству или группе устройств. Например, запускать в виртуальной машине вычисления на виртуальных графических ускорителях, которые будут, в свою очередь, отдавать задачи обсчета реальным графическим ускорителям где-нибудь в облаке.

Помимо перечисленных выше применений, виртуальные устройства предоставляются некоторыми компаниями в режиме PaaS – Platform as Service или "Платформа как услуга что позволяет другим разработчикам испльзовать их для удаленного тестирования своих приложений в разных окружениях [1] [2].

1.3 Создание эмуляторов

Создание эмуляторов – трудоемкий и затратный процесс. Помимо перечисленных ранее типов эмуляторов особой популярностью пользуются так же эмуляторы игровых приставок, как старых, так и новых. Их создание помогает сохранить игры и программы для будущих поколений, формируя целые виртуальные библиотеки, как, например "Internet Archive"[3]. С ростом вычислительной мощности усложняется и архитектура современных игровых приставок. Эмулятор для PlayStation 4 вышел только через восемь лет после выхода самой консоли [4], и то он не может запустить всю коллекцию игр, а PlayStation 4 не принадлежит последнему поколению приставок.

Создание эмуляторов какого-либо аппаратного обеспечения укладывается в следующих этапах:

Первый этап является проектировочным и его результат зависит от нужд конкретной реализации, временных ограничений, соображений о целесообразности. Второй этап тоже, до некоторой степени, проектировочный, но уже здесь можно принять решение воспользоваться эмуляторами, которые поддерживают

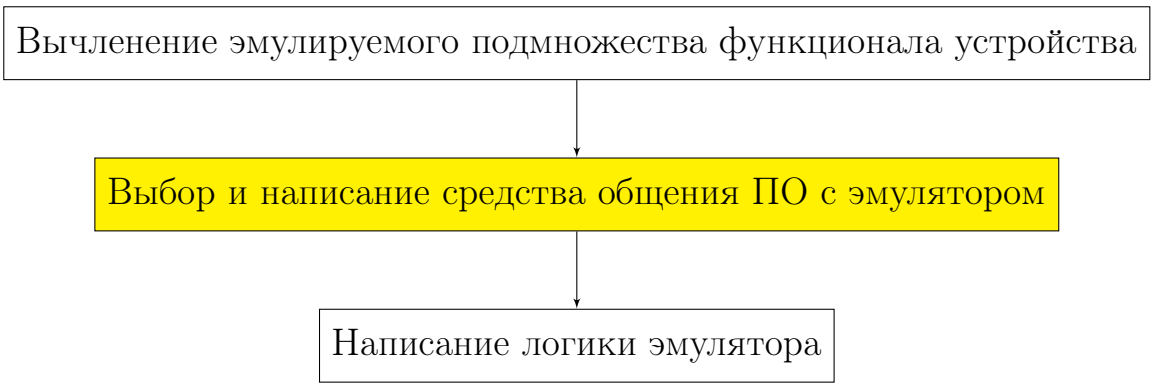

Рисунок 1.2 — Этапы создания эмулятора аппаратного обеспечения

встраивание виртуального аппаратного обеспечения, что автоматически решит задачу. Третий этап является полностью практическим.

Облегчить второй и третий этап можно, если решить использовать уже имеющуюся инфраструктуру эмулятора, встроив в нее эмулируемое аппаратное обеспечение. Использование имеющейся инфраструктуры задает строгий интерфейс встраивания, что ограничивает и тривиализирует общение ПО с виртуальным устройством. Помимо этого, интерфейс встраивания позволяет автоматически генерировать если не всю, то часть взаимодействия между виртуальным устройством и эмулятором.

Используя данный подход, задача создания эмулятора аппаратного обеспечения трансформируется в следующие этапы:

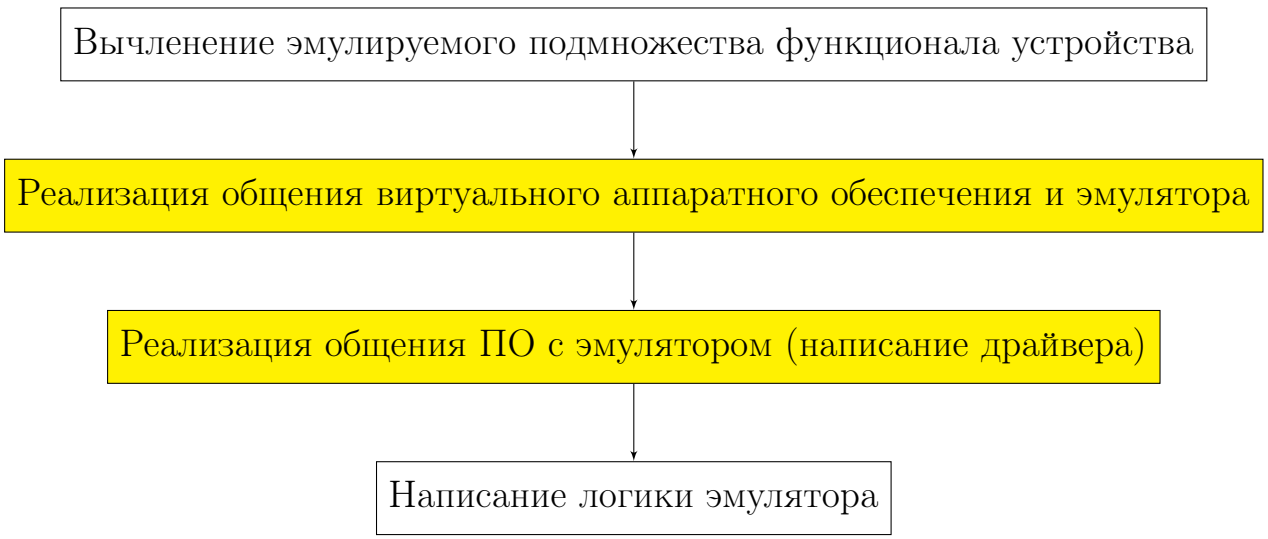

Рисунок 1.3 — Этапы создания эмулятора аппаратного обеспечения с использованием существующего эмулятора

Не смотря на увеличение количества этапов, второй и третий этап данной схемы реализовать проще и целесообразнее, чем второй этап предыдущей схе-

мы, так как во первом случае протокол общения прикладного ПО и аппаратного обеспечения будет находиться частично в эмуляторе аппаратного обеспечения, частично в прикладном ПО. Тогда как во втором случае протокол это реализации драйвера устройства, который в любом случае придется писать.

Но даже в таком подходе можно облегчить второй и четвертый этапы, для этого не хватает только генератора, который мог бы самостоятельно встроить будущее виртуальное аппаратное обеспечение в эмулятор, основываясь на спецификации устройства. Существование такого генератора не только полностью бы убрало второй этап, но и облегчило последний. В случае, если есть возможность получить описание работы эмулируемого аппаратного обеспечения на HDL языке, то четвертый этап создания эмулятора тоже автоматизируется, так как логика работы аппаратного обеспечения уже описана на HDL языке.

1.4 Симуляция HDL

HDL или Harwdare Description Language – специализированный язык для описания структуры и поведния электронных схем. Самыми известными и популярными представителями являются Verilog и VHDL. Данный тип языков помогает разрабатывать электронные схемы на более высоком уровне абстракции, чем раньше, что просто необходимо при проектировании сложных современных интегральных схем.

Процесс превращения HDL-кода в соединения логических вентилей называется логическим синтезом. Результат логического синтеза – список соединений, который может быть зашит на ПЛИС, или исполнен в симуляторе, что позволяет протестировать HDL-программу не используя ПЛИС.

Так как в списке соединений заложена логика работы устройства, пусть и на элементном уровне, то его можно использовать для эмуляции работы устройства.

1.5 Унифицированный подход к созданию эмуляторов аппаратного обеспечения

Унифицированный подход к созданию эмуляторов аппаратного обеспечения позволяет повысить степень автоматизации и корректности рутинной работы по внедрению эмулятора в исполняемую среду. Но сначала надо выбрать эмулятор, который будет использоваться для эмулирования целевого аппаратного обеспечения.

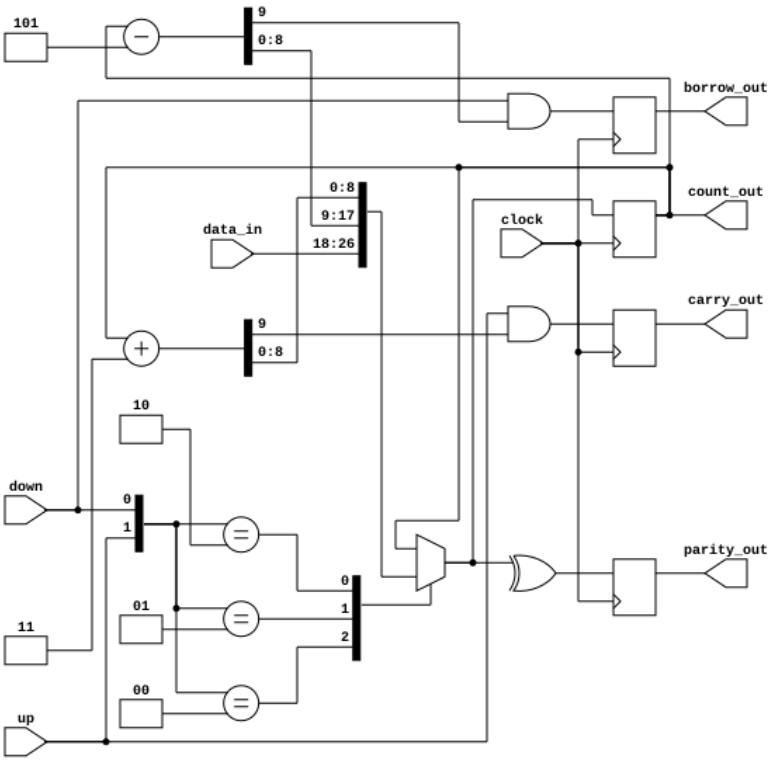

Рисунок 1.4 — Визуализация результата логического синтеза

1.5.1 Сравнение эмуляторов с открытым исходным кодом

Активно развивающихся эмуляторов с открытым исходным кодом мало, Bochs и QEMU единственные, которые можно было бы использовать для создания на их основе эмуляторов аппаратного обеспечения.

Название эмулятора / Свойства	Bochs	QEMU
Кроссплатформенность	Да	Да
Поддержка платформ кроме x86	Нет	Да
Скорость эмуляции	Низкая	Высокая
Возможность добавить собственное устройство	Нет удобного интерфейса	Есть

Таблица 1.2 — Сравнение эмуляторов Bochs и QEMU

Исходя из таблицы 1.2, лучшим эмулятором для поставленной задачи будет QEMU.

1.5.2 Методика добавления целевого аппаратного обеспечения в QEMU

Устройство QEMU

QEMU (Quick Emulator) – эмулятор аппаратного обеспечения различных платформ. QEMU по-умолчанию использует TCG (рисунок 1.5) или Tiny Code Generator (Маленький Генератор Кода) для перевода инструкций эмулируемой системы в инструкции физической машины.

В процессе своей работы TCG разбивает поток инструкций на блоки, разделенные инструкциями ветвления или вызова процедур. В свою очередь, скомпилированные блоки переносятся в кэш трансляции и переиспользуются при следующих вызовах, что ускоряет работу эмулятора. TCG работает только с 32 и 64 битными операндами.

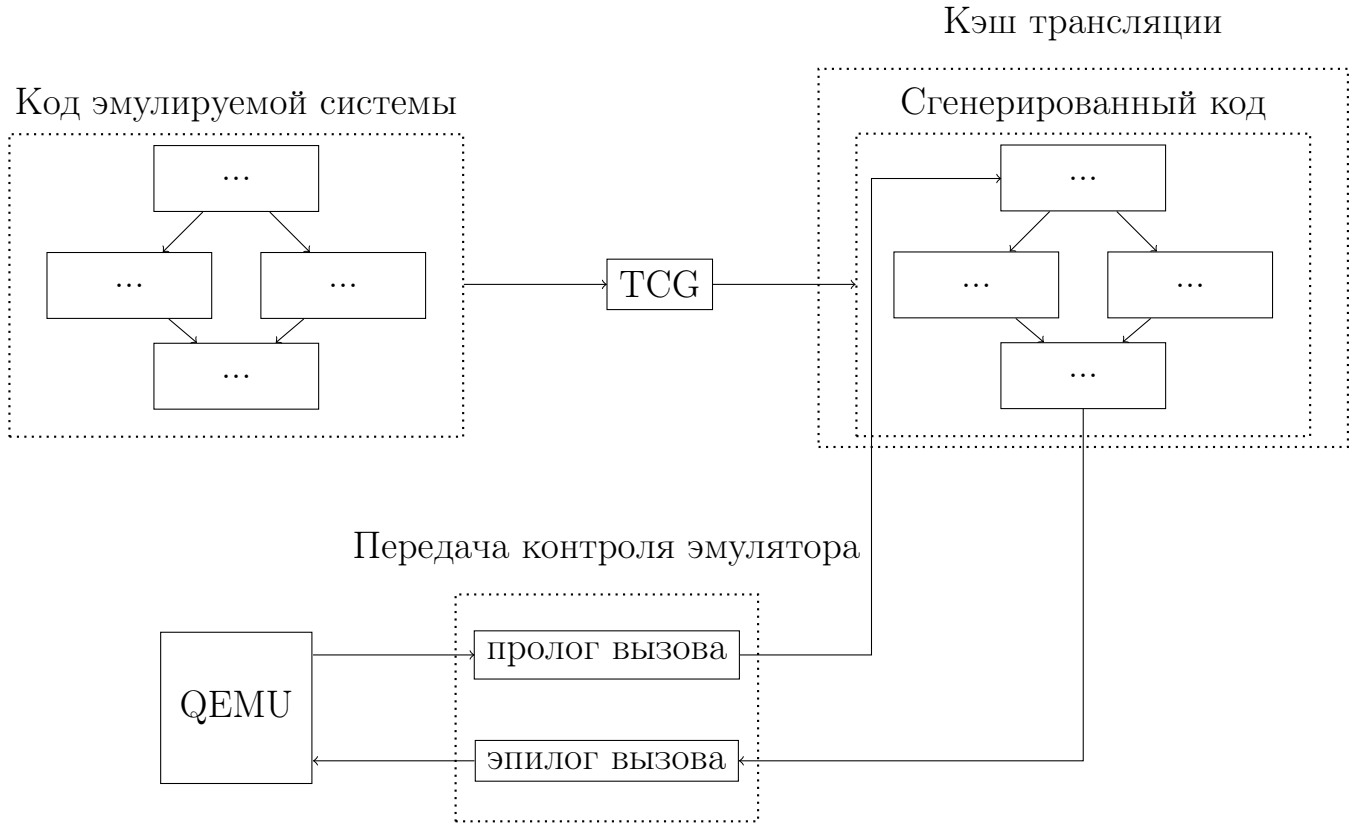

Рисунок 1.5 — Схема работы QEMU TCG

Также TCG применяет оптимизации к компилируемым блокам:

– «долгие» инструкции заменяются на более быстрые альтернативы (если таковые имеются);
– производится анализ времени жизни переменных на уровне блока, при котором удаляются инструкции, не влияющие на вычисления.

Применение TCG позволяет запускать инструкции, предназначенные для любой поддерживаемой архитекуры на любой другой поддерживаемой архитектуре, так как инструкции гостевой системы выполняются на виртуальных процессорах. Это позволяет запускать любые операционные системы и приложения даже на старых процессорах, в которых нет поддержки аппаратной виртуализации, хотя производительность виртуальной машины будет крайне низкая. В случае, когда архитектура эмулируемой и физической системы совпадает, QEMU может применяться как средство виртуализации и использовать аппаратное ускорение. Здесь QEMU может использовать различные ускорители (гипервизоры, рисунок 1.6), отличающиеся в зависимости от операционной системы, в которой эмулятор запущен. Гипервизор используется как «прокладка» для запуска некоторого программного обеспечения в виртуальной среде, скрывая от данного программного обеспечения аппаратное обеспечение машины, на котором это ПО работает.

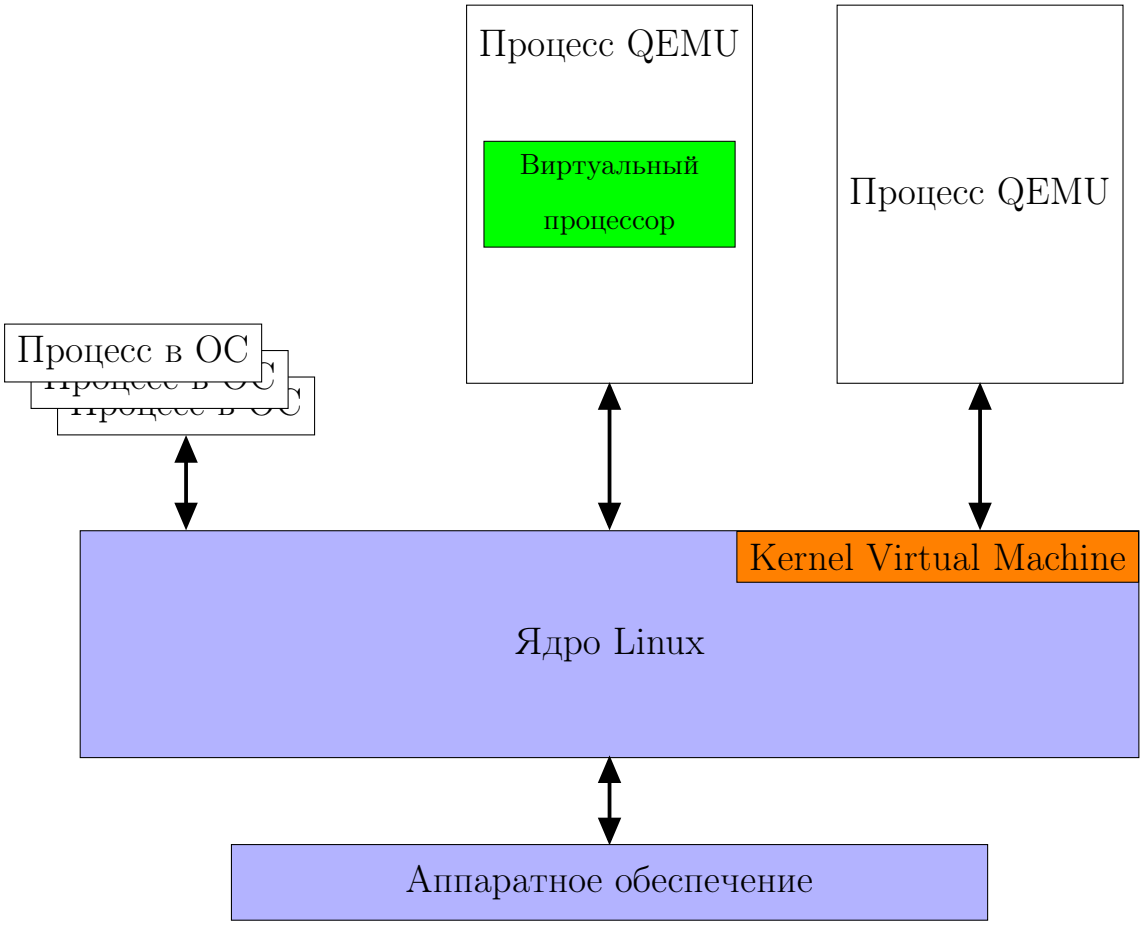

Рисунок 1.6 — Схема работы гипервизора KVM

Ускорители убирают необходимость эмулировать процессор внутри QEMU, а так же позволяют использовать паравиртуализацию (например Virtio [5]) для

устройств ввода-вывода, ускоряя его. При использовании гипервизора, QEMU все-еще должна эмулировать устройства, которые не поддерживаются паравиртуализированными драйверами.

Объектная модель QEMU

Эмулятор QEMU написан в объектно-ориентированной парадигме на языке программирования C [6]. Так как язык C не был создан как объектно-ориентированный язык, QEMU реализует свои механизмы и правила создания, регистрации и управления объектами, их созданием, инициализацией, наследованием и т.д.

В QOM или Qemu Object Model [7], как и в языках, разрабатывавшихся как исконно объектно-ориентированные, каждый созданный тип должен должен быть потомком некоторого другого типа, зарегистрированного в модели. В терминах языка C, QOM-тип описывается двумя структурами:

– структурой класса;
– структурой объекта (т.е инстанцированного класса);

И массивом типа `TypeInfo`, который хранит мета-информацию о типе созданного класса. На рисунке 1.7 показаны отношения между составляющими объектной модели QEMU. Специально написанные макросы позволяют производить действия, присущие объектно-ориентированным языкам, вроде приведения типа объекта к типу родителя.

Встраивание нового устройства в QEMU

QEMU использует QOM для описания устройств. Каждое виртуальное устройство «подключено» к определенной шине. Устройства и шины формируют дерево, в котором каждый узел является либо устройством, либо шиной. Корнем является ‘sysbus‘ «или системная шина»

Для встраивания нового устройства, потребуется определить структуру класса, объекта и массив с описанием типа. Данная работа является рутинной и может быть автоматизирована с помощью кодогенератора.

Написание логики инициализации устройства может быть лишь отчасти автоматизировано, тогда как логики работы автоматизировать не получится, но можно создать дополнительный, скриптовый, интерфейс к C-коду, на котором описание логики устройства будет достаточно быстрым как в смысле работы, так и в смысле временных затрат на кодирование.

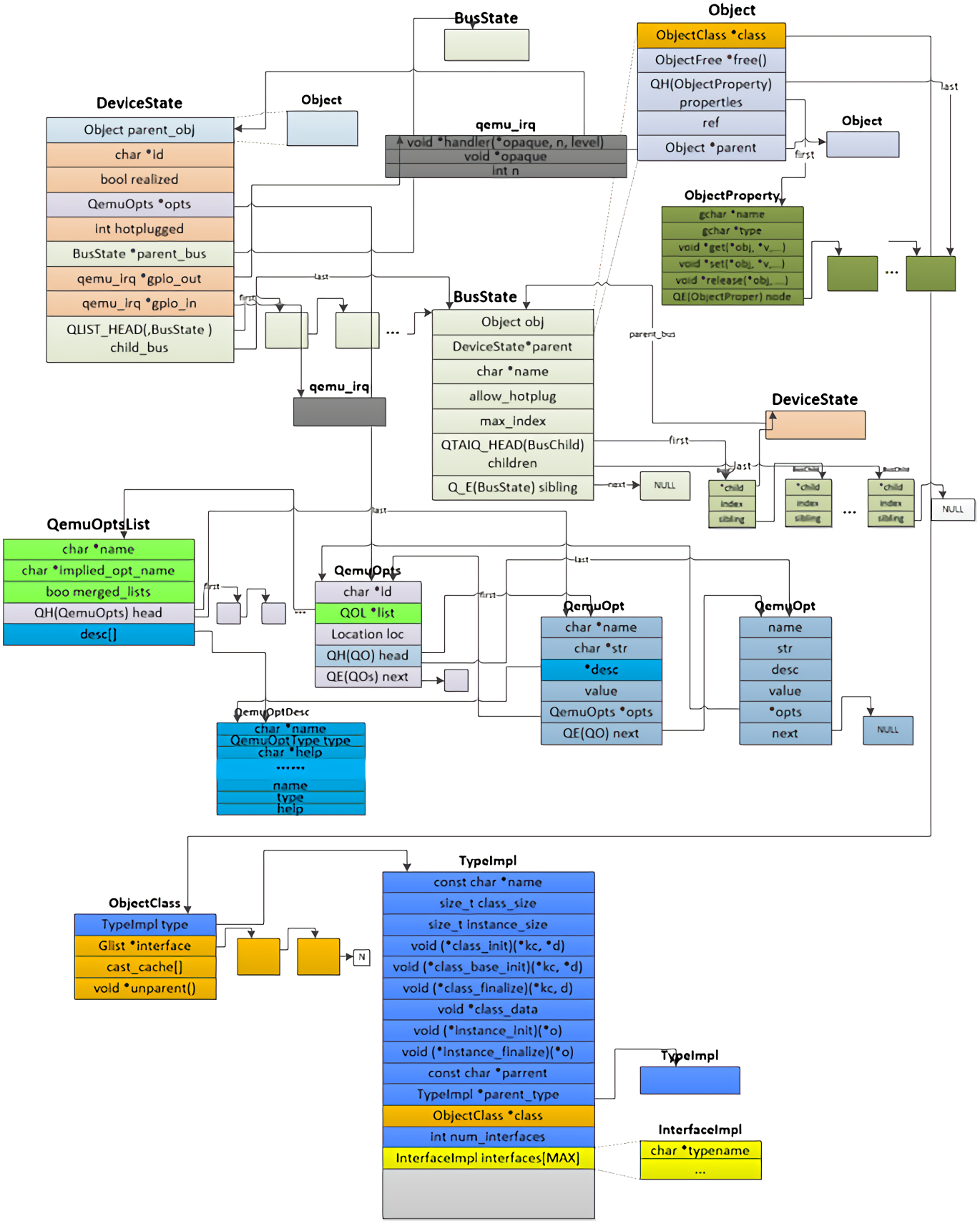

Рисунок 1.7 — Структура QOM

QEMU Machine Protocol

Раздел 2. Формализация процесса создания эмуляторов аппаратного обеспечения

Для того, чтобы решить обозначенные проблемы **??** в создании эмуляторов аппаратного обеспечения, требуется формализовать данную задачу.

2.1 Подход

Формализация и стандартизация Важная задача стандартизации – создание единых норм и правил, действующих на территории разных государств. Воплощение этой концепции в жизнь способствует обмену опытом и технологиями между странами, упрощает проведение операций экспорта и импорта. С 2008 года Россия стала участником Международной организации по стандартизации ISO, членство в которой сегодня имеет 168 стран.

Раздел 3. Программная реализация и экспериментальное подтверждение результатов исследования

Раздел 4. Экспериментальные исследования разработанной методики и алгоритма нахождения НДВ ПО с известной моделью нарушителя

Заключение

Результатом выпускной квалификационной работы стала рабочая версия программного модуля анализа программ на языках C/C++ на недекларированные возможности. позволил унифицировать и ускорил процесс исследования программного обеспечения на НДВ. Уменьшение фрагментации программ по анализу ПО на наличие НДВ позволяет не тратить время программистов на написание анализатора под конкретный продукт, что положительно сказывается на продуктивности всей команды разработчиков.

В рамках выпускной квалификационной работы были решены задачи:

1) исследование предметной области ;
2) сравнительный анализ существующих программных решений;
3) выбор языка и среды разработки;
4) разработка схемы данных ;
5) разработка схемы алгоритма ;
6) программирование ;
7) отладка и тестирование ;
8) разработка документации к .

В заключение автор выражает благодарность и большую признательность научному руководителю Кононовой Александре Игоревне за поддержку, помощь, обсуждение результатов и научное руководство.

Список литературы

1. Lambdatest [Текст]. — URL: https://www.lambdatest.com/android-emulator-online.
2. Genymotion [Текст]. — URL: https://www.genymotion.com/.
3. Console Living Room [Текст]. — URL: https : / / archive . org / details / consolelivingroom.
4. *Кривов, Д.* У PS4 спустя восемь лет работы появился рабочий эмулятор. Но есть нюанс [Текст] / Д. Кривов. — 09.2021. — URL: https://ixbt.games/news/2021/09/11/u-ps4-spustya-vosem-let-raboty-poyavilsya-rabocii-emulyator-no-est-nyuans.html.
5. Virtio: Paravirtualized drivers for kvm/Linux [Текст]. — URL: https://www.linux-kvm.org/page/Virtio.
6. *Schreiner, A.* Object-Oriented Programming With ANSI-C [Текст] / A. Schreiner. — Axel T. Schreiner / Lulu, 2011.
7. Talking about the object system of QEMU [Текст]. — URL: https : / / programmersought.com/article/58151740179.