МИНОБРНАУКИ РОССИИ
Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет «Московский институт электронной техники»

Исследование и разработка методики и алгоритма генерации виртуального аппаратного обеспечения по спецификации

Диссертация на соискание степени магистра по направлению 09.04.04«Программная инженерия»

Научный руководитель: канд. техн. наук, доц. Кононова Александра Игоревна
Соискатель: магистрант. гр. ПИН-22М Уманский Александр Александрович

Москва, 2022

Проблемная ситуация

При создании прикладного ПО для специализированного аппаратного обеспечения дорого обеспечивать разработчиков самим аппаратным обеспечением.

Причины сложившейся ситуации:

- печать экземпляров аппаратного обеспечения в условиях санкций и дефицита полупроводников стала дорогой;
- простаивание программистов, пока происходит печать и доставка аппаратного обеспечения;
- трудоемкость создания собственного виртуального аппаратного обеспечения.

Пример специализированного аппаратного обеспечения

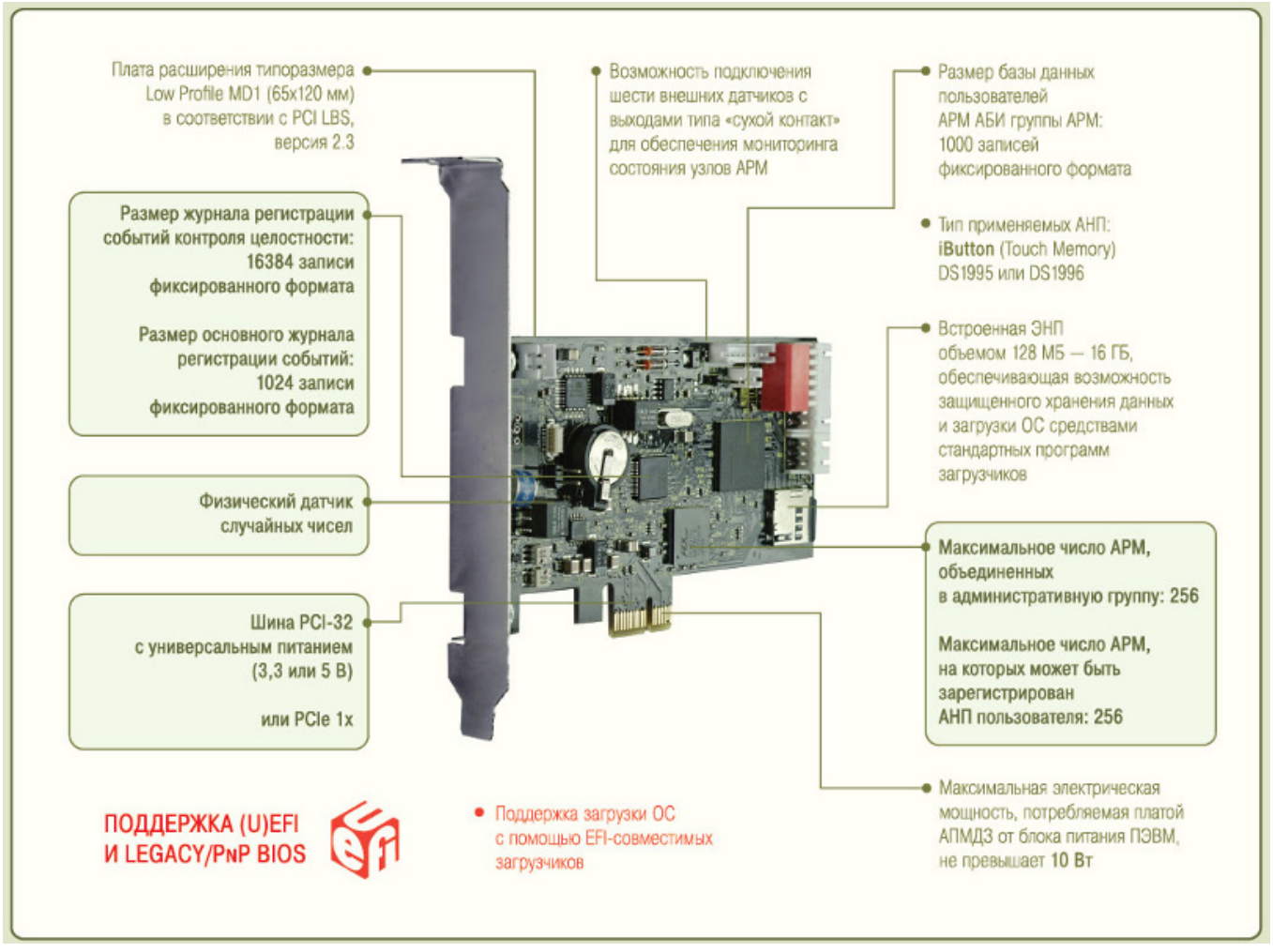


Рис. 1: Аппаратно-программный модуль доверенной загрузки Максим-М1

Цель и задачи диссертации

Цель: снижение трудоемкости создания виртуальных устройств.
Задачи:

- аналитический обзор существующих методов создания виртуального аппаратного обеспечения;
- формализация задачи;
- создание методики и алгоритма генерации виртуального аппаратного обеспечения на основе его спецификации;
- разработка лингвистического аппарата (семантика, синтаксис) языка для создания программ по генерации виртуального аппаратного обеспечения.

Положения, выносимые на защиту

- формализованное представление алгоритма генерации виртуального аппаратного обеспечения;
- лингвистический аппарат (синтаксис, семантика) языка для создания программ по генерации виртуального аппаратного обеспечения;
- экспериментальные результаты применения генератора аппаратного обеспечения.

Анализ существующих методов создания виртуального аппаратного обеспечения

Метод	Особенности	Недостатки
Создание stub-симулятора	Требует создания интерфейсов-адапторов в прикладном ПО	Приходится создавать интерфейсы-адапторы для каждого разрабатываемого ПО
Использование записи работы аппаратного обеспечения	Быстрый метод, не требует специальных знаний о внутреннем устройстве аппаратного обеспечения	• Взаимодействие ПО с аппаратным обеспечением ограничивается заранее записанными сценариями • Количество записей очень быстро разрастается • Зачастую записи снимаются только с корректных сценариев использования
Использование эмулятора QEMU	• Готовая инфраструктура для создания виртуального аппаратного обеспечения • Постоянная поддержка эмулятора силами сообщества	• Необходимость написания виртуального аппаратного обеспечения на низкоуровневом языке • Необходимость обучения объектной системе QEMU (QOM)

Формализованное представление (грамматика)

⟨*letter*⟩ ::= ‘a’ ... ‘z’ | ‘A’ ... ‘Z’;

⟨*digit*⟩ ::= ‘0’ ... ‘9’ ;

⟨*symbol*⟩ ::= \x20 ... \x7E ; (* любой печатный символ, согласно кодам ASCII *)

⟨*const value*⟩ ::= <digit> | ‘’ { <symbol> } ‘’;

⟨*identifier*⟩ ::= <letter> [{ <letter> | <digit> | ‘_’ }] ;

⟨*block start*⟩ ::= ‘’;

⟨*block end*⟩ ::= ‘’;

⟨*field*⟩ ::= <identifier> ‘=’ <identifier> | <block> ;

⟨*block*⟩ ::= <block start> <field> [{ ‘,’ <field> }] <block end>;

⟨*device definition*⟩ ::= ’#’ <identifier>;

⟨*device class inheritance*⟩ ::= ‘(’ <identifier> ‘:’ <identifier> [{ ‘,’ <identifier> }] ‘)’;

⟨*device class block*⟩ ::= <device class inheritance> <block>;

⟨*bind block*⟩ ::= ‘@bind’ <block>;

⟨*python block*⟩ ::= ‘@py’ <block>;

⟨*program*⟩ ::= <device definition> <device class block> <bind block> <python block>;

Рис. 2: Расширенная форма Бэкуса-Наура генератора виртуального аппаратного обеспечения

Формализованное представление (семантика) I

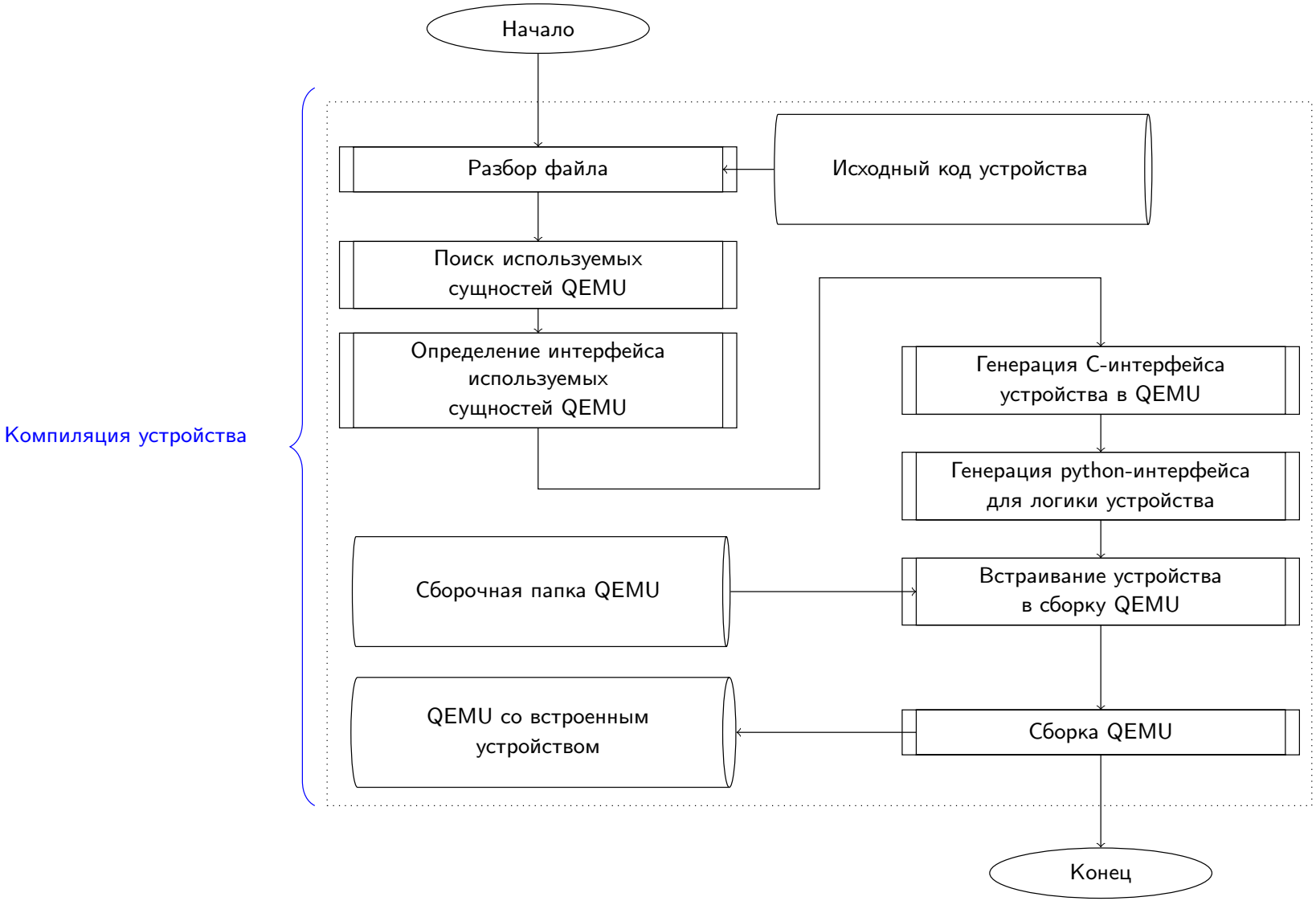
Таблица 2: Денотационная семантика QPyDev

Математическое описание	Значение
$[[assignment]](x,y) = \lambda x.y$	Операция присваивания значения y переменной x
$[[terminate]](m) =$ Завершение работы компилятора	Терминирование компилятора с сообщением m
$[[if]](c,e_1,e_2) = \begin{cases} e_1, & \text{Если } c = true \\ e_2, & \text{Если } c \neq true \end{cases}$	Условное исполнение. Если условие c истинно, то выполняется e_1, иначе e_2
$[[throw\ error]](c,e) = if(c, e_g, terminate)$	Создание и бросание исключения при ложном условии c
$[[lookup]](o) = [[throw\ error]](o \in Q, o)$	Поиск объекта o в множестве объектов QEMU Q. В случае, если объект не найден генерируется исключение

Формализованное представление (семантика) II

$[[< device\ definition >]](i) = lookup(i)$	Поиск указнного класса устройства в объектах QEMU
$[[< device\ class\ inheritance >]](i_1,...,i_n) = lookup(i_1) \wedge ... \wedge lookup(i_n)$	Поск указанного класса для наследования и интерфейсов в объектах QEMU. Для успешного завершения должны быть найдены все объекты
$[[< field >]](v_1, v_2) = [[throw\ error]](v_1 \in Q \wedge v_2 \in C \cup Q, assignment(v_1, v_2))$	Присваивание полям значений при условии, что v_1 принадлежит множеству объектов QEMU, а v_2 множеству констант или множеству объектов QEMU
$[[< block >]](f_1,...,f_n) = field(f_1) \wedge ... \wedge field(f_n)$	Присваивание связанных с одной сущностью полей
$[[< pythonblock >]](b) = assignment(B,B)$	Инициализация специального поля с Python-логикой

Методика создания виртуального аппаратного обеспечения



Программная реализация

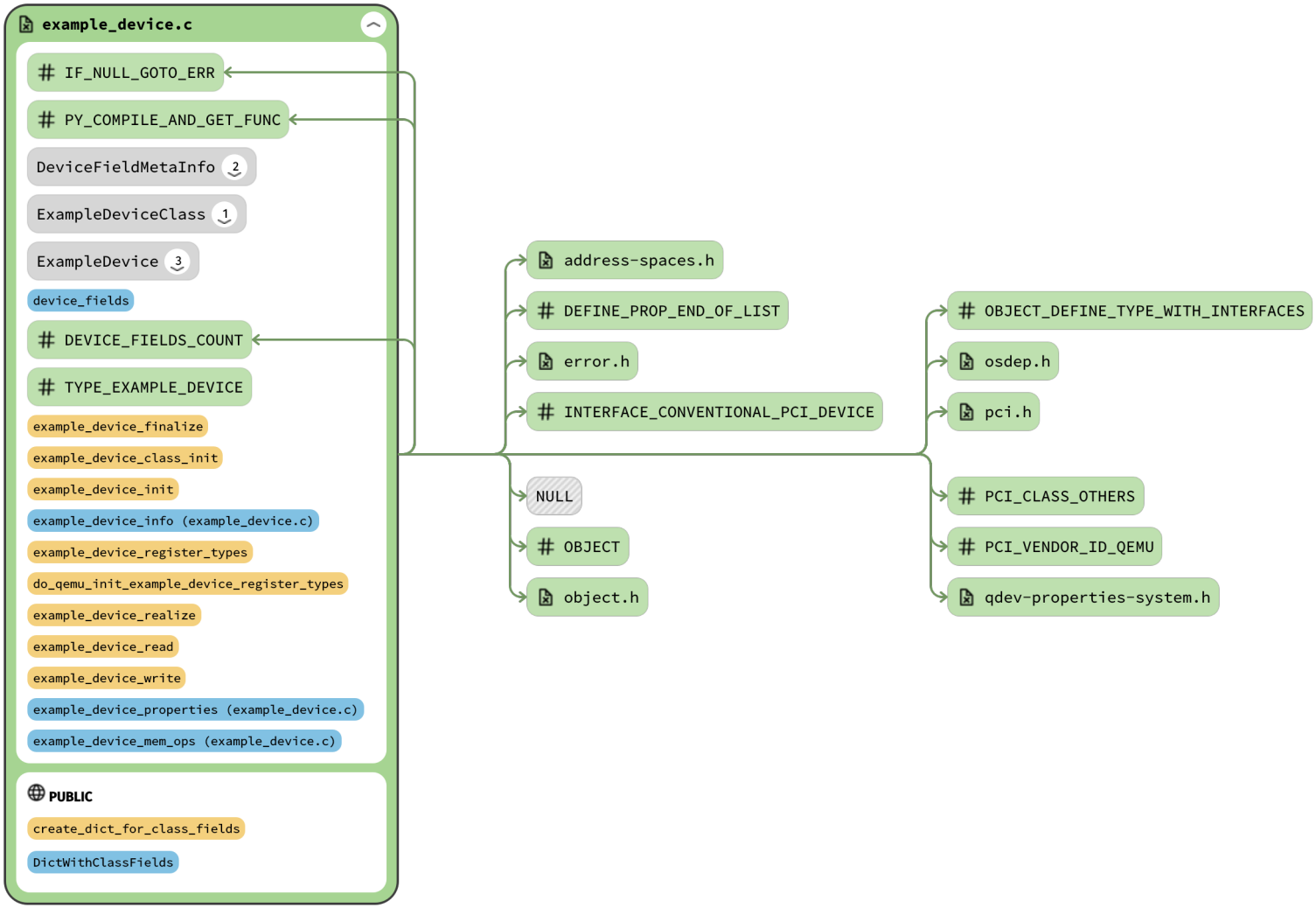


Рис. 3: Пример созданного с помощью QPyDev виртуального устройства

Выбор метрики оценки эффективности

Основные метрики эффективности:

- время разработки виртуального аппаратного обеспечения (в человеко-часах);
- быстродействие сгенерированного виртуального аппаратного обеспечения.

Экспериментальное устройство выполняет задачу сжатия JPEG-картинки. Данная задача легко поддается измерению, так как:

- легко выбрать сложность входных данных – это размер изображения;
- возможна векторизация этапов алгоритма;
- возможно добавить разные подходы к обработке изображения:
  - вызов подпрограммы;
  - отправка данных по сети;
  - реализация алгоритма устройства.

Оценка эффективности

Таблица 3: Сравнение эффективности разработки и производительности виртуальных устройств реализующих алгоритм сжатия JPEG картинки

Метрика	Разработка с нуля		Использование библиотеки	
	C устройство	Python устройство	C устройство	Python устройство
Время разработки в человеко-часах	100	50	35	10
Время сжатия (сек.)	3.915	18.548	1.871	2.786

Вывод: разработка C-устройства дольше, и, соответственно, дороже, но преимуществом является быстрота его работы. Устройство, созданное QPyDev сокращает время разработки вдвое. Использование библиотеки радикально сокращает время разработки в обоих случаях: в 2.8 для C-устройства и 5 раз для Python-устройства. При реализации устройств без сторонних библиотек, Python-устройство в 4.7 раза медленнее аналогичного C-устройства. При использовании библиотек, разрыв сокращается до 1.5 раза, что является более чем приемлимым.

Основные результаты диссертационной работы

- проведен аналитический обзор существующих методов создания виртуального аппаратного обеспечения;
- созданы методика и алгоритм генерации виртуального аппаратного обеспечения на основе его спецификации;
- разработан лингвистический аппарат (семантика, синтаксис) языка для создания программ по генерации виртуального аппаратного обеспечения;

Спасибо за внимание!