УДК 004.94: 621.3.049 DOI: 10.31114/2078-7707-2020-4-112-117

Расширяемый фреймворк для отладки и проверки алгоритмов этапа топологического проектирования

Д.А. Булах, А.В. Коршунов

Национальный исследовательский университет МИЭТ, г. Москва,

dima@pkims.ru, korshun@gmail.com

Аннотация — **Постоянное усложнение задач, требующих решения на этапе топологического проектирования, а также непрерывный рост их количества требуют от разработчиков САПР наличия удобного инструментария для разработки, отладки и сравнения эффективности топологических алгоритмов. Важную роль подобные средства имеют также и для академической среды, поскольку их применение позволит акцентировать внимание в первую очередь именно на разработке алгоритмов, возложив на плечи инструментария такие второстепенные задачи, как чтение и сохранение данных в требуемых форматах, обработка слоёв и прочие. В данной работе предлагается разрабатываемый авторами расширяемый фреймворк для программирования, тестирования и сравнения алгоритмов, разрабатываемых для решения задач на этапе топологического проектирования.**

Ключевые слова — **топология, фреймворк, алгоритм.**

I. Введение

Для поддержания интереса к проблеме проектирования ИС среди студентов высших учебных заведений по всему миру проводится большое число различных конкурсов, направленных на поддержание и развитие алгоритмической базы САПР ИС. Наиболее популярными конкурсами, несомненно, являются конкурсы, проводимые в рамках конференции ICCAD и ISPD [1] - [3]. Как показывают подобные конкурсы, наибольший интерес представляют задачи, связанные с этапом топологического проектирования.

Основной трудностью при решении студентами подобных задач является то, что в университетах студентам в достаточной степени излагаются алгоритмические аспекты решения задач и ими прорешивается большое число примеров, однако практически не раскрываются вопросы, связанные с форматом данных и алгоритмами чтения/записи результатов.

При разработке заданий для подобных конкурсов зачастую встаёт вопрос: что делать с чтением входных данных? Для конкурсов типа ICCAD и ISPD входными данными являются данные в форматах GDSII [4] и LEF/DEF [5]. Для чтения входных данных в этих форматах при выполнении заданий конкурсов используются библиотеки, доступные в исходных кодах и размещённые на сервисе github, либо написанные самостоятельно командами-участниками соревнований.

Для студентов российских вузов практика использования находящихся в открытом доступе библиотек для работы с реальными производственными форматами не является общепринятой, в связи с чем при организации отечественных конкурсов и заданий приходится в большинстве случаев использовать упрощённые входные форматы. Например, известный популяризатор конкурсов в области алгоритмов САПР Юрий Панчул признаётся, говоря о задаче чтения входных данных: «После обсуждения со студентами мы решили парсирование верилога заменить на упрощенный ввод координат ячеек из текстового файла» [6].

Подобный подход неудобен, поскольку он отдаляет людей, решающих задачи, от реальной ситуации, заставляя работать с искусственными данными.

Идеальным решением является использование библиотеки или ряда библиотек, позволяющих осуществлять чтение, запись и базовую обработку реальных форматов файловых данных, применяемых при проектировании ИС, предоставляя разработчику алгоритмов возможность работать уже со считанными данными в удобном формате.

В данной работе представлен разрабатываемый нами фреймворк LEAF (англ. Layout Educational Algorithmic Framework), позволяющий в графическом виде из отдельных элементов формировать маршрут обработки топологии ИС, и предоставляющий возможность дополнять перечень элементов пользовательским кодом.

Под термином «фреймворк» мы понимаем архитектуру программного обеспечения, которая даёт разработчикам возможности:

- пользоваться базовым встроенным функционалом рассматриваемого программного обеспечения, предоставляющим доступ к технологическим форматам файлов посредством внутренних функций для их чтения и записи;
- расширять функциональность программного обеспечения путём написания пользовательских модулей и алгоритмов;
- выстраивать, моделировать и верифицировать маршрут обработки топологий с использованием как встроенных, так и пользовательских модулей.

II. Внутреннее устройство фреймворка

A. *Работа с фреймворком и графический интерфейс*

На рис. 1 показан основной графический интерфейс пользователя. В программе, помимо общепринятых элементов управления (меню, панель инструментов и строка статуса) есть три основных поля:

1. список доступных для размещения элементов, из которых можно строить маршрут;
2. рабочая область, в которой размещаются элементы маршрута;
3. окно вывода сообщений, статусов операций и прочей текстовой информации.

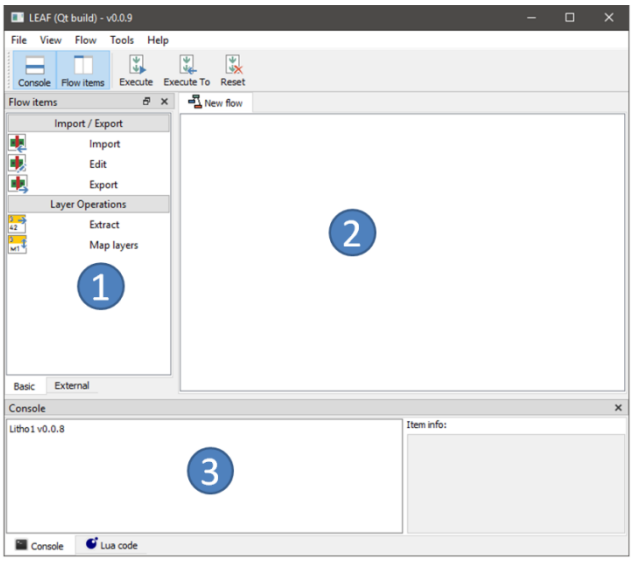

Рис. 1. Интерфейс фреймворка LEAF

Процесс составления и выполнения маршрута в предлагаемом фреймворке состоит в размещении предлагаемых пользователю элементов маршрута (рис. 1, область 1) в рабочем поле (рис. 1, область 2). Графически проводя связи между элементами маршрута, у разработчика есть возможность показать, в какой последовательности будут выполняться действия. Пример тестового маршрута показан на рис. 2.

В рамках этого маршрута происходит чтение двух топологий GDSII (элементы маршрута «INPUT»), для первой топологии производится выбор слоёв, с которыми в дальнейшем будет происходить работа (элемент маршрута «EXTRACT»), затем выполняется операция объединения всех элементов из разных топологий (элемент маршрута «AND»), после чего результат сохраняется в файл GDSII.

Для работы пользователю доступны как встроенные элементы маршрута, расположенные на вкладке Basic, так и пользовательские (при их наличии), расположенные на вкладке External.

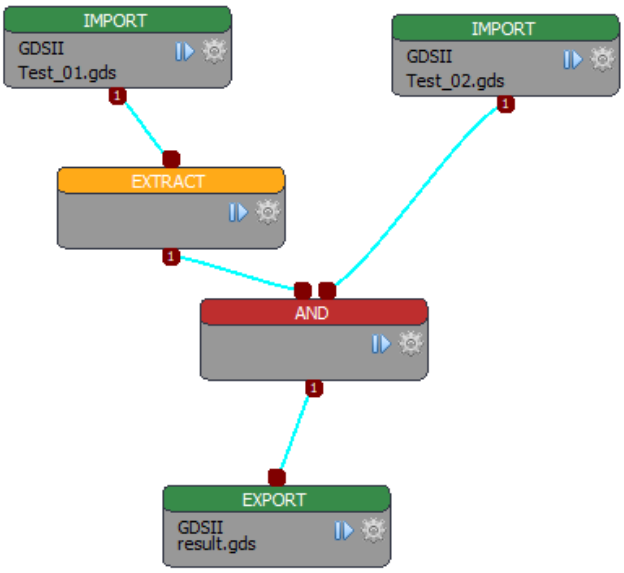

Рис. 2. Пример формирования тестового маршрута

Основная идея представляемого фреймворка заключается как раз в возможности создания пользовательских элементов маршрута. Подробнее принцип работы фреймворка рассмотрен в подразделе B, а пример разработки своего элемента маршрута – в разделе III.

B. *Общая архитектура*

Фреймворк построен по модульной архитектуре (рис. 3). Он включает в себя базовую и пользовательскую части.

Базовая часть состоит из каркаса - графического оконного приложения, разработанного на языке программирования C++ с использованием кроссплатформенной библиотеки Qt, и набора базовых элементов, позволяющих реализовывать такие операции маршрута, как ввод/вывод данных, экстракцию данных и некоторые другие.

Пользовательская часть формируется из перечня разработанных пользователями модулей, выполненных в виде динамически подключаемых библиотек.

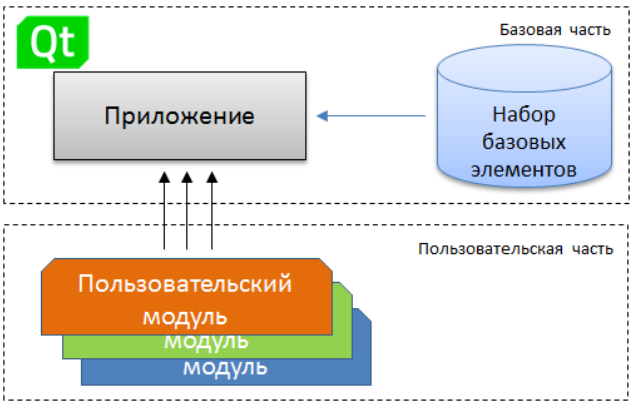

Рис. 3. Архитектура фреймворка

Такой подход позволяет:

- реализовывать код разрабатываемых алгоритмов на любом компилируемом языке программирования, с которым работает разработчик;
- не отвлекаться на задачи чтения и сохранения результатов в применяемые технологические форматы, уделяя всё внимание разработке алгоритмов;
- выполнять сравнение времени выполнения каждого отдельно взятого модуля в пределах платформы;
- свободно перемещать разработанные модули между различными платформами и операционными системами; единственное, что потребуется – перекомпиляция модуля, никаких изменений в программный код вносить не придётся;
- в перспективе реализовать доступную онлайн коллекцию пользовательских модулей, разработанных различными группами разработчиков и предоставленных для общего пользования.

Пользовательские модули разрабатываются на компилируемом языке программирования и собираются в виде динамически подключаемых библиотек (англ. Dynamically Linked Libraries, DLL). При размещении их в заранее определённой директории элементы автоматически распознаются при запуске фреймворка и могут быть доступны для использования в маршруте.

Такой подход позволяет, не меняя программный код, использовать представленную архитектуру на любой платформе. На данный момент работа фреймворка проверена под ОС Windows и Linux, идёт тестирование под Mac OS. Конечно, в случае использования разных операционных систем придётся перекомпилировать программный код как самого фреймворка, так и пользовательских библиотек, однако тот факт, что при разработке используется широко распространённая кросплатформенная библиотека Qt сводит все неудобства к минимуму.

C. Набор стандартных компонентов

На данный момент в базовой поставке фреймворка реализован следующий перечень стандартных модулей маршрута:

- загрузка описания топологии из файла (элемент «Import»);
- сохранение результата - топологии, получившейся в результате прохождения разрабатываемого маршрута, в файл (элемент «Export»);
- ручной ввод топологии в редакторе (элемент «Edit»);
- назначение топологических слоёв (англ. mapping, элемент «Map Layers»);
- экстракции отдельных топологических слоёв и элементов топологии (элемент «Extract»).

Для импорта и экспорта топологии на данном этапе развития проекта реализована возможность чтения и записи данных в форматах GDSII (бинарный и текстовый).

Ручной ввод топологии возможен только при наличии отдельно установленного топологического редактора. По умолчанию фреймворк настроен на работу с открытым кроссплатформенным топологическим редактором KLayout [7]. В текущей версии фреймворка свой собственный редактор и просмотрщик топологии отсутствуют. Для своего запуска многие топологические редакторы используют дополнительные ключи командной строки (помимо имени файла), поддержка всех этих возможностей в фреймворке реализована.

Как известно, формат GDSII не хранит технологические имена топологических слоёв, оперируя только численными значениями. Для удобства оперирования слоями в фреймворке реализована поддержка ручного мэппинга слоёв.

Разработка многих алгоритмов предполагает обработку не всей топологии, а только определённых её слоёв. Также интерес может представлять обработка выборочных топологических структур из всех доступных в файле топологии. Для комфортной работы разработчиков в базовый состав фреймворка входит элемент маршрута для экстракции требуемых данных из топологии.

D. Внутренние структуры данных

Разрабатывая код элемента маршрута, программист должен наследоваться от базового класса, имеющего в своём распоряжении ряд виртуальных методов, переопределяя которые можно управлять поведением элемента.

Упрощённая UML диаграмма интерфейсного класса приведена на рис.4.

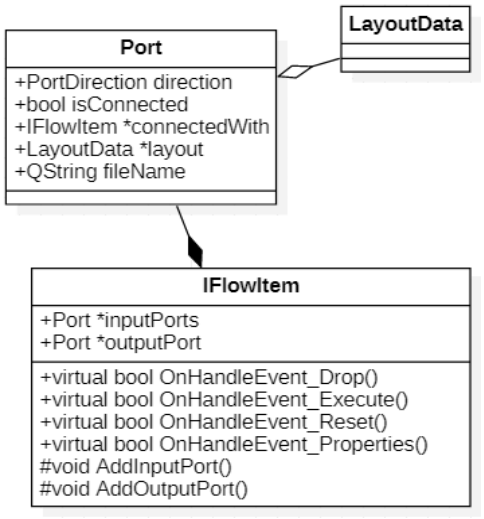

Рис. 4. UML диаграмма, показывающая взаимное отношение типов данных

Функции, начинающиеся на OnHandleEvent_* позволяют переопределить обработчики различных событий:

- Drop – элемент попадает в рабочее поле путём перетаскивания;
- Execute – элемент должен исполниться;
- Reset – должны сброситься все проведённые вычисления;
- Properties – должен вызваться диалог для переконфигурирования свойств элемента маршрута.

Имея возможность обратиться к входным портам (переменная inputPorts), разработчик может получить доступ либо к имени файла с результатом, получаемым из предыдущего элемента маршрута (который соединён выходом с соответствующим входом текущего элемента), либо непосредственно к считанным данным. Управляется это поведение вызовами соответствующих инициализирующих функций в конструкторе. Аналогично осуществляется работа с выходными данными (переменная outputPorts).

III. Пример разработки модуля

Пример создания класса будет проще сего показать на конкретной задаче. Поскольку сложность решаемой задачи в данном случае не принципиальна, более того, в рамках статьи на простой задаче будет проще продемонстрировать принцип разработки модуля, в качестве задачи выбрана трассировка алгоритмом Ли топологии с препятствиями и двумя терминалами.

A. *Конструктор объекта*

В конструкторе класса разработчик указывает основные свойства своего модуля. Пример кода конструктора приведён в листинге 1.

```
FlowItem_Lee::FlowItem_Lee()
: FlowItem(fit_user) {
  AddInputPort();
  AddOutputPort();
  SetTitle(QString("LEE ROUTER"));

  EnableTimeMeasure(ma_drop, false);
  EnableTimeMeasure(ma_execute, true);
  EnableTimeMeasure(ma_reset, false);

  SetDataAt(pd_input, dm_layout);
  SetDataAt(pd_output, dm_layout);
}
```

Листинг 1. Код конструктора

Вызов функций AddInputPort() и AddOutputPort() говорит фреймворку о том, что у разрабатываемого элемента маршрута будет один вход и один выход.

Функция EnableTimeMeasure принимает два аргумента. Первый задаёт обработчик, для которого устанавливается или снимается необходимость измерения времени выполнения, второй – собственно сама информация о том, нужно или нет измерять время работы модуля.

Вызов двух последних функций определяет то, в каком виде элемент маршрута принимает данные от предыдущего элемента (параметр pd_input) и в каком виде передаёт следующему (параметр pd_output). Передаваемое значение dm_layout говорит о том, что на момент вызова обработчика сообщения OnHandleEvent_Execute фреймворком из входного файла должна быть прочитана топология и размещена во внутренних структурах данных.

B. *Реализация обработчика выполнения элемента*

Учитываем, что в конструкторе было задано, что на момент выполнения элемента, а точнее на момент вызова функции OnHandleEvent_Execute, при попадании в этот метод можно обратиться к полю layout входного порта (рис. 4).

Обращаясь к этому полю у разработчика есть возможность получить информацию о входной топологии. Архитектура класса LayoutData отражает структуру данных, хранящуюся в файле формата GDSII, поддерживаются как бинарная, так и текстовая версия формата.

Архитектурно это реализовано благодаря применению наследования, и схематично показано на рис. 5.

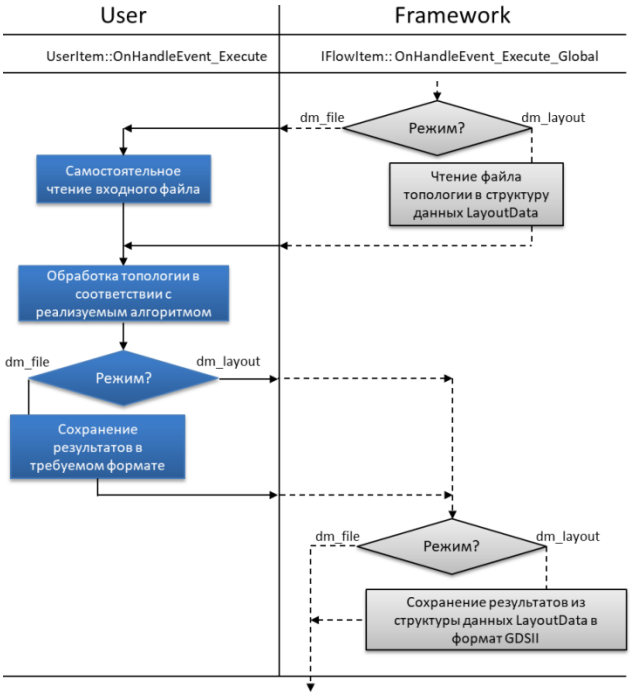

Рис. 5. Взаимодействие функций чтения/записи результатов в пределах реализации механизма наследования и виртуализации

В момент «выполнения» элемента маршрута вызывается функция OnHandleEvent_Execute_Global базового класса, в которой выполняется проверка того, в каком виде ожидаются входные данные. Если данные ожидаются в виде имени файла, управление

немедленно передаётся в обработчик выполнения OnHandleEvent_Execute разрабатываемого класса. В противном случае, в базовом классе происходит считывание данных в структуру типа LayoutData, отражающую содержимое файла топологии, и только после этого происходит передача управления в наследуемый класс.

При возвращении исполняемого кода в вызывающую функцию базового класса будет проведена проверка, нужно ли выполнять автоматическое сохранение данных в файл, поэтому, если разработчик выбрал работу в режиме самостоятельного сохранения результатов работы алгоритма, он должен позаботиться о сохранении всех данных в выходном файле самостоятельно. В противном случае, он должен позаботиться о сохранении результатов изменений в структуре данных LayoutData, и сохранением займётся функция наследуемого класса.

C. Проверка результатов работы

В рамках фреймворка реализована возможность не только запускать элементы маршрута на выполнение, но и просматривать результаты их выполнения. В системе такая возможность реализована автоматически, разработчику не нужно заботиться о том, как это выполнить.

Согласно внутреннему представлению, каждый элемент содержит информацию, показывающую его статус: вычисления ещё не были проведены или вычисления проведены. Разница в статусе показывается графически (рис.6.).

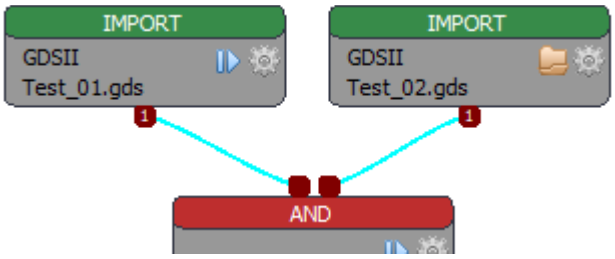

Рис. 6. Графическая иллюстрация статусов элементов: «не запускался» (слева), «запускался, есть результаты» (справа)

Если состояние соответствует значению «не запускался», при нажатии на кнопку запуска происходит вызов и выполнение кода обработчика OnHandleEvent_Drop соответствующего элемента. Пример такого элемента показан на рис.6 слева. На рис. 6 справа показан пример элемента, для которого уже был выполнен запуск, его статус «результаты готовы», в результате чего результат становится доступен для просмотра (отображается иконкой другого вида). При нажатии на этот элемент открывается связанная программа просмотра топологии, в которой отобразится результат работы элемента.Помимо демонстрации результатов, сам процесс обработки топологических данных может быть отображён в консоли программы (рис.1, область 3). В консоли может быть размещена текстовая информация любого содержания.

IV. Пример использования разработанного модуля

Для демонстрации принципа работы алгоритма была создана топология, показанная в редакторе KLayout на рис. 7.

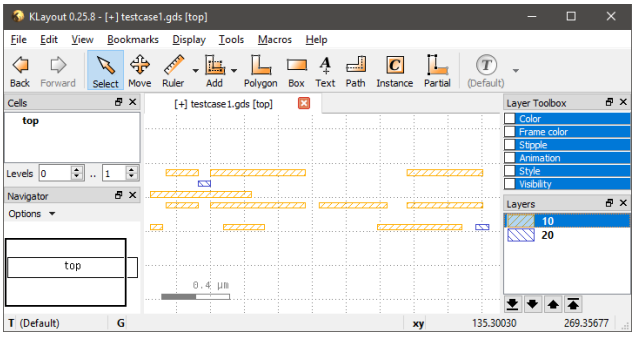

Рис. 7. Входные данные

Слой 10 содержит ряд элементов, представляющих собой препятствия. Слой 20 содержит 2 элемента – гипотетические терминалы, которые нужно соединить. Информация о том, какие данные расположены на каком слое считается известной, с точки зрения функционирования фреймворка это не является принципиальным моментом.

Маршрут, составленный в рассматриваемой программе с использованием разработанного в предыдущем разделе элемента маршрута, показан на рис. 8.

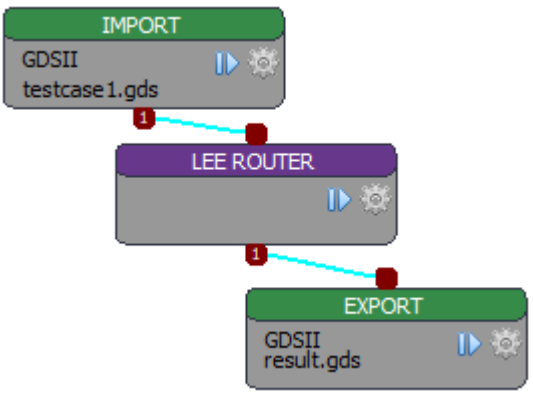

Рис. 8. Сформированный графический маршрут

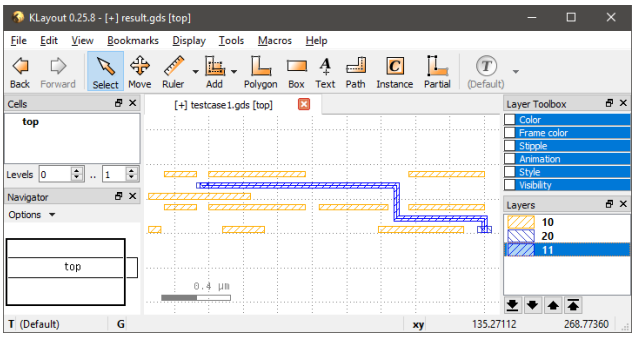

Рис. 9. Выходные данные

Результат работы реализованного алгоритма, сохранённый в выходном результирующем файле, показан на рис. 9.

Добавленное межсоединение, соединяющее два терминала со слоя 20, размещается на слое 11. Его номер задаётся внутри программного кода пользовательского элемента маршрута.

V. Результаты и выводы

Представлены архитектура и принцип работы фреймворка для разработки алгоритмов на этапе топологического проектирования. Разработанный интерфейс пользователя, основанный на построении последовательности и взаимном соединении графических элементов, даёт возможность строить маршрут, позволяя проводить верификацию на каждом шаге маршрута. Применение фреймворка в образовательном процессе позволит акцентировать внимание на разработке и отладке алгоритмов, работая при этом реальными, применяемыми в производстве, форматами файлов.

В дальнейшем планируется дополнить код проекта, добавив во встроенный модуль импорта возможность считывать данные в форматах LEF/DEF, а также добавить возможность проводить сравнение двух топологий (элемент маршрута «Diff»), что позволит сравнивать результаты выполнения различных элементов маршрута, автоматически оценивая их результативность.

Поддержка

Работа выполняется в рамках госзадания МИЭТ (тема 0719-2020-0017 / FSMR-2020-0017).

Литература

[1] M. Kim, S. Huang, R. Lin and S. Nakatake, "Overview of the 2017 CAD contest at ICCAD: Invited paper," 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Irvine, CA, 2017, pp. 855-856, doi: 10.1109/ICCAD.2017.8203867.

[2] U. Schlichtmann, S. Das, I. Lin and M. P. Lin, "Overview of 2019 CAD Contest at ICCAD," 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Westminster, CO, USA, 2019, pp. 1-2, doi: 10.1109/ICCAD45719.2019.8942133.

[3] URL: http://www.ispd.cc/?page=contests (дата обращения: 25.06.2020).

[4] URL: https://cad-contest-2016.el.cycu.edu.tw/CAD-contest-at-ICCAD2016/pdf/Problem_C.pdf (дата обращения: 25.06.2020).

[5] URL: http://www.ispd.cc/contests/18/index.html (дата обращения: 25.06.2020).

[6] Юрий Панчул. Трассировка silicon-а в формате хакатона. Без Physical Design не будет Айфона. URL: https://habr.com/ru/post/500300/ (дата обращения: 25.06.2020).

[7] URL: https://www.klayout.de/intro.html (дата обращения: 25.06.2020)

An Extensible Framework for Developing and Testing of the Layout Processing Algorithms

D.A. Bulakh, A.V. Korshunov

National Research University of Electronic Technology, Moscow

dima@pkims.ru, korshun@gmail.com

***Abstract* — Growing complication of physical design tasks and its number continuous growth requires CAD developers to have convenient tools to design, implement and merge the results of physical design algorithms. It also very important to have such tools in education since its application in educational process will allow to focus on algorithm developments and implementation and to delegate secondary tasks (input data reading/writing, layers processing etc.) to a such tool. Preseting this paper we introduce framework for physical design algorithms studying, development, implementation and debugging. Funding: The reported study is performed within MIET project number FMR-2020-0017.**

***Keywords* — layout, framework, algorithm.**

References

[1] M. Kim, S. Huang, R. Lin and S. Nakatake, "Overview of the 2017 CAD contest at ICCAD: Invited paper," 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Irvine, CA, 2017, pp. 855-856, doi: 10.1109/ICCAD.2017.8203867.

[2] U. Schlichtmann, S. Das, I. Lin and M. P. Lin, "Overview of 2019 CAD Contest at ICCAD," 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Westminster, CO, USA, 2019, pp. 1-2, doi: 10.1109/ICCAD45719.2019.8942133.

[3] URL: http://www.ispd.cc/?page=contests (access date: 02.04.2020).

[4] URL: https://cad-contest-2016.el.cycu.edu.tw/CAD-contest-at-ICCAD2016/pdf/Problem_C.pdf (access date: 02.04.2020).

[5] URL: http://www.ispd.cc/contests/18/index.html (access date: 02.04.2020).

[6] Yuri Panchul. Trassirovka silicon-a v formate hakatona. Bez Physical Design ne budet iphona (Silicon routing for hackaton. No iPhone without a phisical design) . URL: https://habr.com/ru/post/500300/ (access date: 02.04.2020).

[7] URL: https://www.klayout.de/intro.html (access date: 25.06.2020).