# An Academic Framework for IC Physical Design Algorithms Development

Dmitry Bulakh, Andrey Korshunov
*Department of IC design*
*National Research University of Electronic Technology*
Zelenograd, Moscow, Russian Federation
dima@pkims.ru, korshun@gmail.com

Anton Datsuk
*IHP - Leibniz-Institut für innovative Mikroelektronik*
Frankfurt (Oder), Germany
datsuk@ihp-microelectronics.com

***Abstract*—Analyzing electronic design automation (EDA) computer aided design (CAD) contests it can be seen that the dominant part of tasks and problems belongs to physical design. This is due to the increments of the complexity of semiconductor devices that discovers an increasing number of new problems to be solved in nanometer scale layouts. As the number of the tasks and their complexity grows the new EDA tools should enhanced to solve the main algorithmic tasks. In this paper we present a new extensible EDA framework for physical level design algorithms development, testing and implementation. The framework provides ready-made solutions for secondary, auxiliary tasks such as file I/O and layout data manipulation. Developed EDA framework was integrated in educational process and made it possible to develop physical design algorithms using technological files.**

**The reported study was done with a support of the state assignment of MIET (Theme No. FSMR-2020-0017/AAAA-A20-120071490038-5)**

***Keywords—Electronic Design Automation, Computer Aided Design, Physical Design, IC Layout***

## I. Introduction

Integrated circuits (IC) design more and more requires the use of a computer-aided automation. This leads to development of new electronic design automation software tools, which take into account new physical effects, utilize modern computing capabilities, process large volumes of data. At the same time, some parts of the software, mostly service procedures and functional blocks, that provide main algorithms function preparation, remain unchanged.

Development of the algorithms is related to the physical design that requires many service procedures of this kind. For example, functional blocks are responsible for layout data import and export/import from/to files, functional blocks for various kinds of layout data processing (cells and layers manipulation) and so on.

Speaking about educational process in high education institutions, we often meet a situation when the use of real input files (e.g., technological files in binary and bulky text formats, for example GDSII and LEF/DEF) are ignored due to their complexity in favor of simpler textual file formats (rephrase). These file formats have much easier internal structure, usually contain only minimal required set of data and are easier to read, write and understand. The main idea of this approach is to avoid wasting time for understanding the file format, thus spending more time for implementation and debug of algorithms.

The developed algorithms are designed to work fine on a small and simple example input files but cannot be run on real layouts with all their specifics and much bigger volumes of data.

Many CAD contests give as the input data files in real technological formats [1-2]. The files can be processed based on several open-source libraries, which provide an access to binary data and technological layout formats. However, all of these libraries work with different data structures and do not provide ability to read different file formats into a single data structure. This complicates usage of these data for the algorithms. In other words, the implementation of open-source libraries binds programmers to only one input file format and blocks the capability to apply these libraries for multi-format file reading.

The optimal solution is to develop a library that supports a functionality to process different formats and to store layout information in a single data structure.

In this paper, we developed graphical PDLab (Physical Design Laboratory) framework that helps to solve the aforementioned problems.

By framework, we mean the following:

- a possibility of usage the basic built-in modules and blocks for layout data manipulation which are not related directly to the developing layout processing algorithms (for example modules for reading and writing binary files, different kinds of layout processing and others);

  • an extension of the basic functionality of a framework by development of external modules that can communicate with both embedded and user modules, provide layout data transfer between the modules and so on;

- an opportunity to build an «algorithmic flow» to either process or verify physical designs and to control the results at each step of the flow.

## II. Framework Internals

### A. *Using framework to build a flow*

The framework consists of a graphical user interface (GUI) application and a set of libraries for development of user-defined modules using the embedded API. The main interface of PDLab framework is shown in Fig.1.

The interface provides four main work areas:

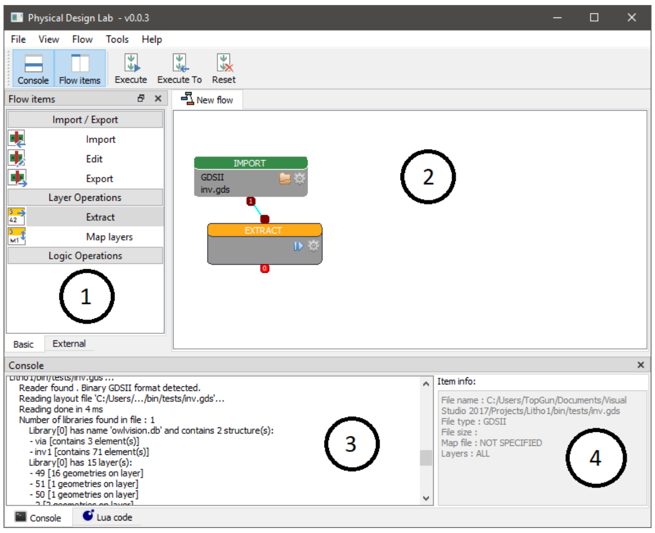
- a list of available integrated basic modules (or flow items) (1), these modules can be placed and connected in main working area to build an algorithmic flow;
- main working area (2);
- a log area console to display the application messages with the support of Lua scripting language to control configuration of PDLab (3);

- an information window (4) to display information about selected flow item.

The flow items window (1) contains a tab widget to display both the basic modules provided with the application and the user-defined ones. The user modules are supposed to be implemented as a dynamic-link library. The provided dynamic-link library exports specific data described below.

To build an algorithm flow it is needed to place all of the required flow items from the «Flow items» window into main working area and then to connect them in correct sequence. An example of a flow is shown in Fig.2.

This figure shows a flow performing logic AND operation on a selected layer. The layer geometry information can be extracted from different layout sources. First layout was taken from input file «Test_01.gds» (operation performed by flow item «Import»). Second layout was edited manually using external layout editor (operation performed by flow item «Edit»). From the layout that was red several layers extracted (operation performed by flow item «Extract»). The result of this operation is saved to the output file «result.gds». In this example we assume that extracted layer from the left branch of the flow has the same identifier (name or number) as the edited one from the right branch. The logical AND operation is implemented by merging geometries from both layout sources.



```
Litho1/bin/tests/inv.gds ...
  Reader found . Binary GDSII format detected.
  Reading layout file 'C:/Users/.../bin/tests/inv.gds'...
  Reading done in 4 ms
  Number of libraries found in file : 1
    Library[0] has name 'owlvision.db' and contains 2 structure(s):
    - via [contains 3 element(s)]
    - inv1 [contains 71 element(s)]
    Library[0] has 15 layer(s):
    - 49 [16 geometries on layer]
    - 51 [1 geometries on layer]
    - 50 [1 geometries on layer]
```

Item info:

```
File name : C:/Users/TopGun/Documents/Visual
Studio 2017/Projects/Litho1/bin/tests/inv.gds
File type : GDSII
File size :
Map file : NOT SPECIFIED
Layers : ALL
```

Fig.1. Physical Design Lab interface

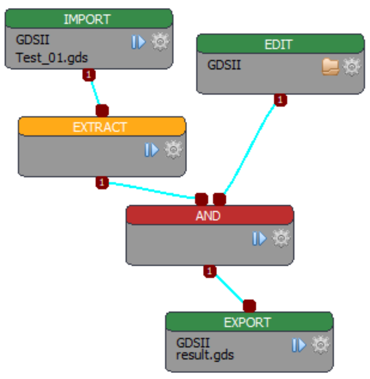


Fig.2. Example flow with build-in I/O flow items, layers manipulation item and logic operation item

## B. *Software architecture*

The developed framework is build using the module architecture approach and it is shown in Fig.3.

The core module of the application is a GUI developed in C++ based on a cross-platform Qt library which is often used for educational projects [3].

The user flow items and the integrated modules are embedded by using an external static library. The libraries contain an implementation of graphical flow item and an access to the layout data.

## C. *Accessing layout data*

Library «Layout.lib» provides an interface and a data structure to read, store and write layout data. It supports reading capability for different layout formats. The implementation guarantees that the extracted layout information from the sources of any kind stores the data in the same structure.

Simplified UML diagram of layout class's relationship is presented in Fig.4.

To read and write layout data the library provides a set of functions that perform import and export operations for layout stored in different formats.

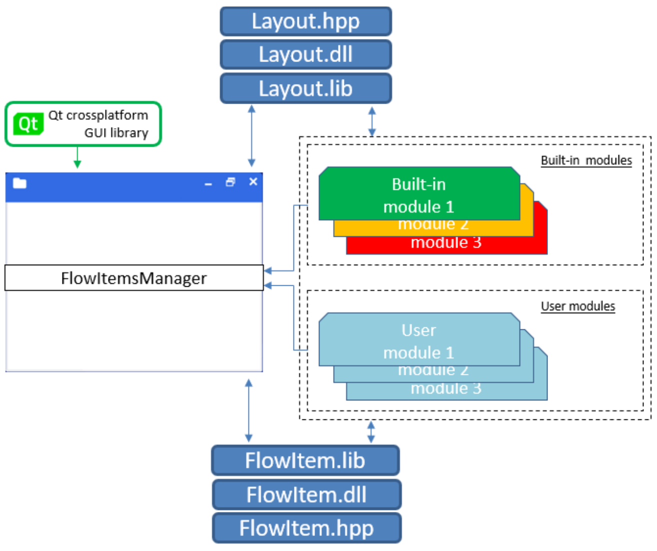


Fig.3. Framework architecture

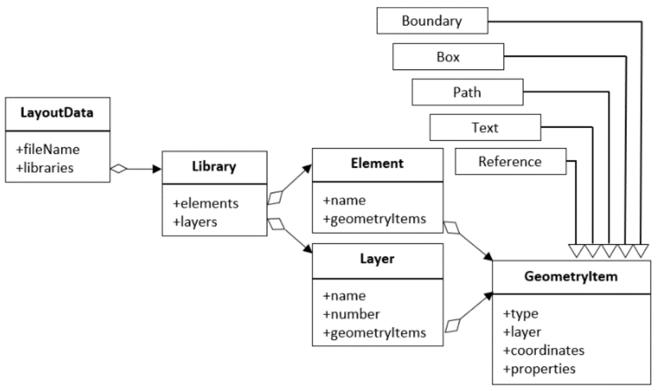


Fig.4. Layout classes UML diagram

### D. *Implementing flow item*

To implement a user-defined flow item a developer implements a new class and inherits it from the base abstract class AbstractFlowItem.

An example of a simplified code inherited from AbstractFlowItem is given in listing 1. There are four mandatory functions provided by the class, which are expected to be overwritten by the user. These functions are callbacks. They are called by PDLab application when either a flow item is placed into the main working area or a user executes or resets a flow item or a user opens its properties dialog.

```
class FlowItem_Import : public AbstractFlowItem {
public:
  bool OnHandleEvent_Drop() override final;
  bool OnHandleEvent_Execute() override final;
  bool OnHandleEvent_Reset() override final;
  bool OnHandleEvent_Properties() override final;
public:
  void GetInfoString(QString &info) override final;
};
```

Listing 1. An example of code inheritance for the «Input» flow item

Each of the flow items may contain input and output pins, which are used to connect the flow item during layout development. A flow items provides additional string areas to display an important information. The «execute flow item» and «open properties dialog» buttons were developed to invoke the flow item and call the properties dialog window respectively. An interface of different type flow items is shown in Fig.5.

Each of the flow items may be executed in either standalone or a complete mode. In case of a standalone mode all the flow item that precede the executed one are invoked. The complete mode executes all flow items placed into the main PDLab interface. These two scenarios can be demonstrated using Fig. 2. In case user executes «AND» flow item then all the preceding items («EXTRACT», «IMPORT» and «EDIT») are also launched. However, the flow item «EXPORT» is not invoked. To execute all items the complete mode needs to be run.

The AbstractFlowItem class contains additional functions to assign input and output ports, information strings and layout data.

To get an access to the information saved after execution in the preceding flow items the input pins are used. The flow graph shown in Fig. 6 demonstrate this use case. The flow item «AND» grands access to the Test_01.gds data of the left «IMPORT» flow item by clicking on the left input pin. To access the layout data information of Test_02.gds file the ritgh pin needs to be selected respectively. The approach helps to manipulate layout data on different steps using various flow item controllers. The data can be also transmitted to the other controllers using the output pins of the flow item module.

Each flow item can have one of two possible states not yet executed (Fig.6, a) and already executed and it has results to display (Fig.6, b). If a flow item has was executed, but the execution failed, its status return to «not yet executed».

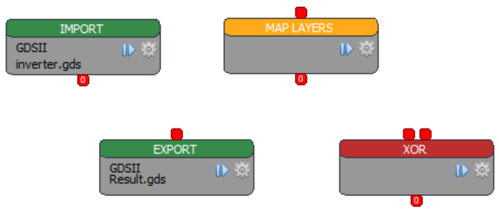


Fig.5. Flow items with only output pins (a), only input pins (b), both types of pins (c) and variable number of input pins(d).

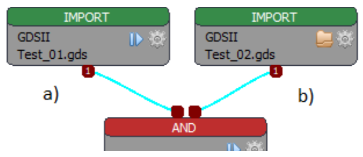


Fig.6. Flow items with different statuses

## III. Layout Processing

This section of the article demonstrates a use case of layout data processing using PDLab software by using XOR2 design example (Fig. 7). An extraction of specific layers set extracted from the GDSII input file is shown using an external freeware layout viewer KLayout [4]. The design flow task to extract a set of metal layers from the GDSII layout file is presented in Fig.8.

When the «Import» flow item is invoked, the «Extract» flow item gets an access to the extracted layout data by selecting its input pin. Then the extracted list of layers from the input file are displayed in the «Extract» options dialog window. On selecting some layers to be extracted the «Extract» flow item could be run. Fig.8 shows that layers 49 and 51 are selected for the extraction.

The extracted layout data are exported in a separate file that is assigned to the «Export» flow item. The result of such flow is illustrated in Fig.9. The processed data could not only be saved, but they could also be used in the next flow operations. This use case is shown in Fig. 10.

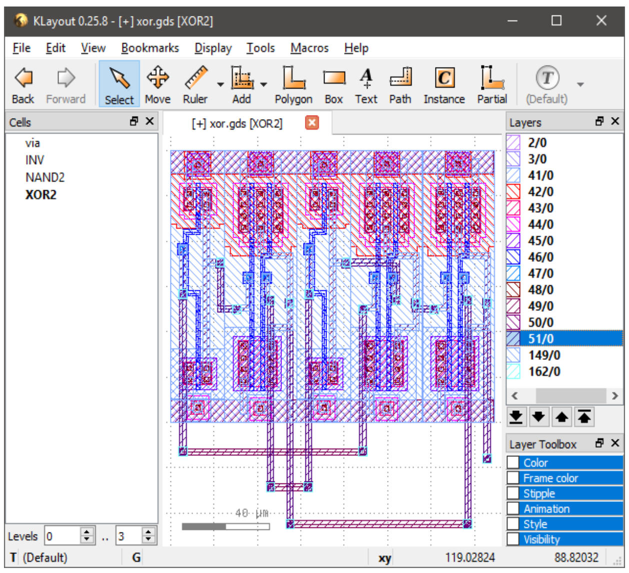
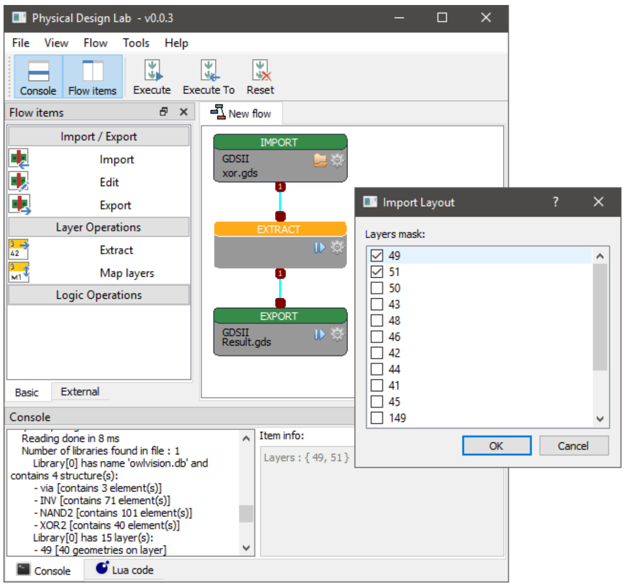


Fig.7. XOR2 GDSII file opened in KLayout



Layers mask:

| | |
|---|---|
| ☑ | 49 |
| ☑ | 51 |
| ☐ | 50 |
| ☐ | 43 |
| ☐ | 48 |
| ☐ | 46 |
| ☐ | 42 |
| ☐ | 44 |
| ☐ | 41 |
| ☐ | 45 |
| ☐ | 149 |

```
   Reading done in 8 ms
   Number of libraries found in file : 1
      Library[0] has name 'owlvision.db' and
contains 4 structure(s):
      - via [contains 3 element(s)]
      - INV [contains 71 element(s)]
      - NAND2 [contains 101 element(s)]
      - XOR2 [contains 40 element(s)]
      Library[0] has 15 layer(s):
      - 49 [40 geometries on layer]
```

Item info:

Layers : { 49, 51 }

Fig.8. PDLab interface with layout layers dialog interface

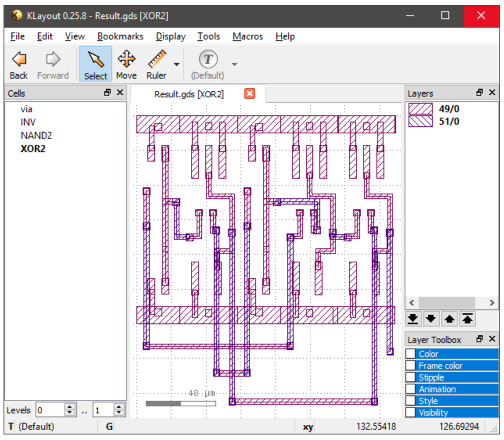


Fig.9. Result layout of a flow

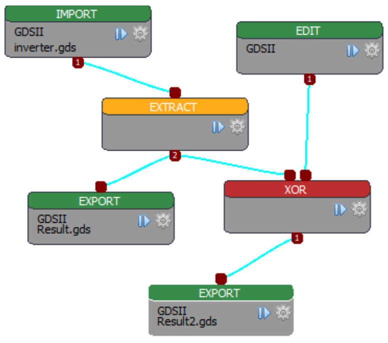


Fig.10. Multiple layout manipulations

## IV. CONCLUSION

The presented software approach simplifies the process of algorithms development and verification applied for IC layout designs.

The solution proved to be useful and important for academic community. It allows getting a visual control and a clear representation of flow steps that are performed for layout processing. The PDLab framework allows to avoid any layout preparation tasks and gives user an opportunity to focus on algorithms development.

## 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] I. Mezei, "Cross-platform GUI for educational microcomputer designed in Qt," 2017 IEEE East-West Design & Test Symposium (EWDTS), Novi Sad, 2017, pp. 1-4, doi: 10.1109/EWDTS.2017.8110109.

[4] KLayout Layout Viewer and Editor «About the Project» page. URL: https://www.klayout.de/intro.html (Access date: 29.01.2021).