# Integrated Development Environment for Visual Parallel Programming

Boris Sedov, Alexey Syschikov, Vera Ivanova
Saint Petersburg State University of Aerospace instrumentation
Saint Petersburg, Russia
{sedov.boris, alexey.syschikov, dashalrn3}@gmail.com

**Abstract**

This paper represents an integrated development environment for visual parallel programing. We believe that parallelism is clearly seen in visual form. Many classes of the problems are naturally described in flowchart form. So we design the integrated develop environment, described all its parts, and trace the entire path of the algorithm's development.

**Index Terms:** integrated development environment, visual parallel programming, flowchart.

## I. Introduction

We are supporters of a visual programming. We want to develop programs in a graphic form. Diagramming programs are not suitable for this purpose because there are no any tools for testing and debugging of an algorithm. So we need to develop the integrated development environment.

The standard approach to solving the problem is:

1. Developing of the algorithm usually in a flowchart representation.
2. Writing a code of this algorithm in some programming language.

In many ways visual representation is better for understanding of the whole algorithm. This is a reason why it is used to design a concept of an algorithm.

For a detailed description a code is more suitable. In a traditional approach these two representations are separated. We propose to combine them in one and use advantages of both of them.

For example, parallelism is clearly seen in visual form than in textual code. Moreover, there is a large class of problems which are naturally described in flowchart approach: signal processing, communication tasks, etc.

The final product of the proposed integrated development environment is an abstract representation of an algorithm. So we will have a universal solution for the task and can import it to various programming tools.

## II. Main part

### *A. Flowchart editor*

The central part of the integrated development environment is the flowchart editor. The development of an algorithm is started here. Blocks and interconnection between

them are built here. Each block can contain sub-flowchart or C code. At any moment designer can switch to any level of the hierarchy, from the level of the common view (a basic flow chart of an algorithm) to the code in any function.

The flowchart editor is designed in such way not only to make the process of building algorithms a quick and convenient, but also to prevent potential errors “by design”. Let’s consider some of them.

A form of nodes indicates that a structure of an algorithm is not completed. Not connected ports are represented as “labels” which are not natural for this scheme, as you can see in fig. 1. It should show to programmer that it’s an inconsistence and force him to fix it.

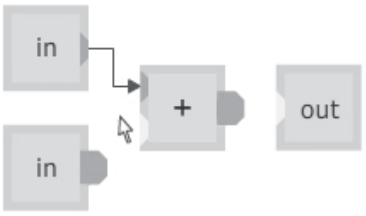


Fig. 1. Construction of an algorithm in the flowchart editor

One of the basic principles of the flowchart editor is to hide unnecessary information. When a scheme is completed, input and output ports are hiden into the nodes bodies (fig. 2, left image). When a designer needs to do some changes, he can switch to the extended view and every node in the scheme will be annotated with numbered ports, which can be captured by the mouse cursor easily.

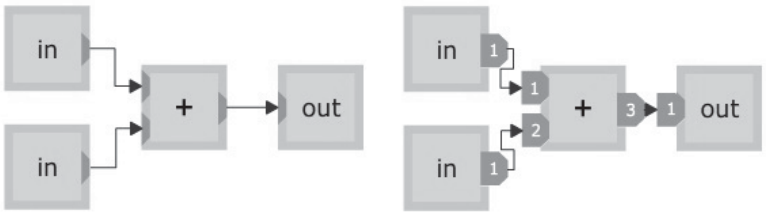


Fig. 2. The scheme without tabs (left) and with tabs (right)

The flowchart editor offers the library of basic complex nodes, such as conditional “IF” operator, “FOR” loop, “WHILE” loop etc. The designer can make a node with any complexity. A complex object has two states: folded and unfolded. When complex object is unfolded, other scheme is obscured, which gives opportunity to focus on the selected node. In the fig. 3 (right image) you can see empty “FOR” loop body with non-connected ports.

If you compare the left and the right images, you will notice that signals from input and output ports of the loop are not visible and a designer needs to folded and unfolded the “FOR” loop to find this information. Of course it is unacceptable, that’s why a special feature was developed. A highlighting of a complex object external connections shows to which nodes input and output ports are connected (fig. 4).

During all process of building an algorithm syntactic correctness is checked dynamically. If semantics of some objects is known, dynamically semantics correctness also is checked.

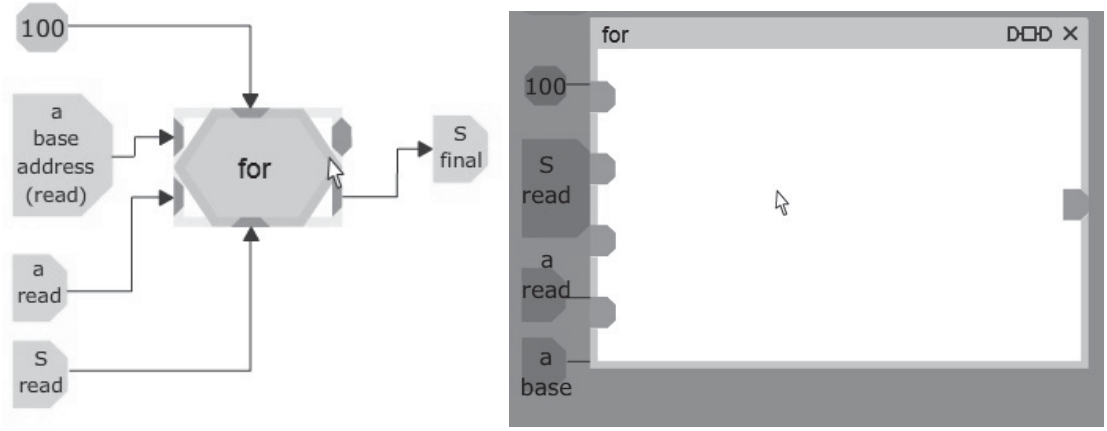


Fig. 3. A folded “FOR” loop with external connections (left), unfolded for loop (right)

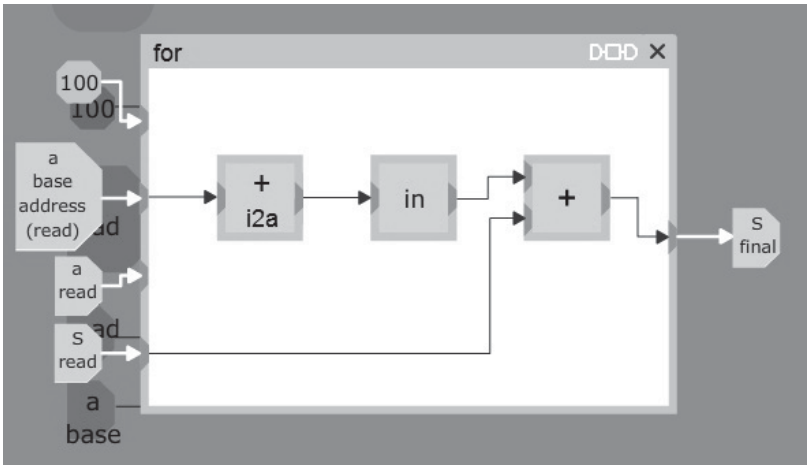


Fig. 4. A highlighting of the loop external connections is turned on

Special debugging tools can be plugged in the development environment.

*B. Debugging*

After the designer has a syntactically and semantically correct scheme of an algorithm, scheme should be debugged for a functional correctness. It can be done by connecting a simulator to the integrated development environment, for example DCNSimulator (Digital Communication Network Simulator) [1]. An abstract representation of an algorithm will be executed in virtual runtime environment (fig. 5).

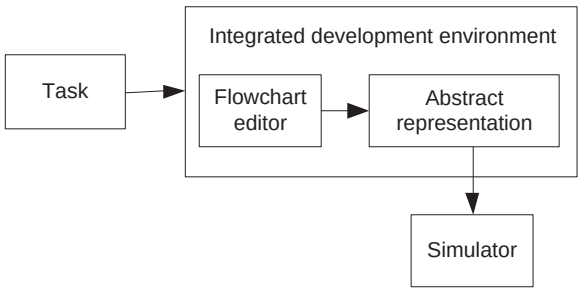


Fig. 5. Connecting simulator to the integrated development environment

### *C. Interactive debugging*

The main advantage of the interactive debugging is that everything that happens in a simulation is reflected in a flowchart. Thus, if during a simulation a problem is detected, a place in an algorithm is shown and it takes less time to fix it.

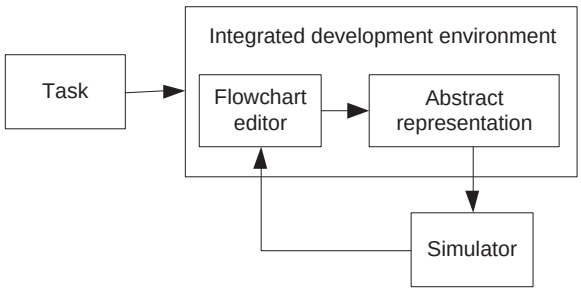


Fig. 6. Interactive debugging

The traditional types of the debugging, step by step, step over, step into, breakpoints are also supported and reflected on the flowchart.

### *D. Result*

As a result we get an intermediate representation of an algorithm. Model of computational [2] guarantee a correctness of an algorithm.

We can connect any module to the development environment, which could recognize the intermediate representation of an algorithm, for example: synthesizer, code generator, compiler, simulator etc. (fig. 7). It is possible to import an intermediate representation to a required programming language by using of a code generator.

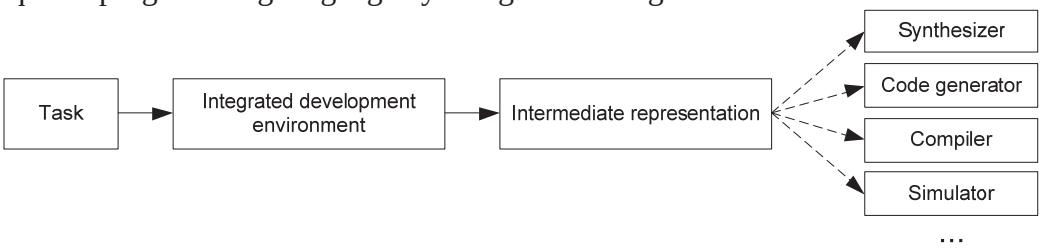


Fig. 7. Plug-ins

## III. CONCLUSION

This paper represents the integrated development environment for a visual parallel programming, which allows designing a solution of a problem in a combination of visual and textual representation, translating it into a verified intermediate representation that can be further processed by different backend and tools.

Integrated development environment is designed to reduce the possibility of errors at every step of a solution development.

Future work will include the flowchart editor with more useful functions to make it more suitable for designers.

## REFERENCES

[1] Sheynin Y.E., Volkov P.L., Onischenko L.V., Razhivin D.B., Cherniy A.S., Eganyan A.V., Nikolsky V.F., Kosyrev S.A., “Software support of the VLSI family “Multicore-designer” for the construction of the parallel

structures and distributed signal processing systems", "Questions of Radio electronics", a series of "Electronic computing equipment (EWT)", issue 3, 2008

[2] V.I. Ivanov, Y.E. Sheynin, A.Y. Syschikov, "Programming model for coarse-grained distributed heterogeneous architecture", XI International Symposium on Problems of Redundancy in Information and Control Systems: Proceedings, SUAI, 2007, c.246-250