Integrated Development Environment for Multi-core Systems

MOMČILO V. KRUNIĆ, RT-RK, Institute for Computer Based Systems LLC, Novi Sad
NENAD B. ČETIĆ, University of Novi Sad, Faculty of Technical Sciences, Novi Sad
MIODRAG M. ĐUKIĆ, University of Novi Sad, Faculty of Technical Sciences, Novi Sad
IVAN Đ POVAŽAN, University of Novi Sad, Faculty of Technical Sciences, Novi Sad
MIROSLAV V. POPOVIĆ, University of Novi Sad, Faculty of Technical Sciences, Novi Sad

Professional paper
UDC: 004.4

Development of the software application that provides comfortable working environment of embedded software applications was always a difficult task to achieve. To reach this goal it was necessary to integrate all specific tools designed for that purpose. This paper describes Integrated Development Environment (IDE) that was developed to meet all specific needs of a software development for the family of multi-core target platforms designed for a digital signal processing in Cirrus Logic Company. Eclipse platform and RCP (Rich Client Platform) was used as a basis, because it provides an extensible plug-in system for customizing the development environment. CLIDE (Cirrus Logic Integrated Development Environment) represent the epilog of that effort, reliable IDE used for development of embedded applications. Validation of the solution is accomplished thru 2641 JUnit tests that validate most of the CLIDE's functionalities. Developed IDE (CLIDE) significantly increases a quality of a software development for multi-core systems and reduces time-to-market, thereby justifying development costs.

Key words: *IDE, multi-core systems, embedded systems, Eclipse, multi-core*

1. INTRODUCTION

Integrated development environment, abbreviated IDE, is a development environment that provides developers a broad set of tools for software development. Integrated development environment integrates various specific tools that make developers jobs much easier. Some of those tools are: text editor sensitive to a programming language of interest, along with standard assembler components, linker, simulator, and components for the controlled execution and debugging.

This paper describes one solution of the IDE for multi-core systems based on the Eclipse RCP platform [1], [2]. Custom Eclipse based development environment is developed by the many companies and manufacturers of embedded target platforms. Some of the most famous are: Texas Instruments [3], Analog Devices [4], Silicon Labs [5], Freescale [6], Renesas [7], Mentor Graphics [8]. Each of these solutions uses the Eclipse RCP platform as the basis on which they build specific parts of the eclipse plug-ins to provide new possibilities to the overall development environment. Flexibility of the Eclipse environment is used as a basis for many similar projects in scope of embedded systems. In [9] the Eclipse is used for development of an open source project IDE Laika for the software of an embedded Linux system targeted for the Nokia 770 Internet Tablet. This is very similar to our approach and applied to similar targets, but still lacking of low level support for the bare-bone applications and assembler language. Our solution gives abstraction for low power classes of embedded devices, where features like register view and peripheral space addresses are key features. Save-IDE [10] is an Integrated Development Environment for building predictable component-based embedded systems which was developed in the Eclipse development platform using the Eclipse Modeling Framework (EMF), the Graphical Modeling Framework (GMF), and the Acceleo plug-ins. Model

Author's address: Momčilo Krunić, RT-RK, Institute for Computer Based Systems LLC, Novi Sad, Narodnog fronta 23a

Paper received: 27.05.2014.
Paper accepted: 15.09.2014.

driven development in GMF framework is used in [10]. On the other hand, we use only GEF for graphical editing support since after some evaluation that we conducted we concluded that GMF stability doesn't offer industrial strength support. Eclipse and RCP platforms are also recognized as a friendly working environment for development of language specific IDE because building IDE from scratch represents a huge effort. An open source project that builds a free Haskell IDE [11] is also based on the Eclipse. Similarly as in [11], we added assembler language support as Eclipse plug-in extension. T-Clipse [12] is an Integrated Development Environment (IDE) that uses an Eclipse for development of Two-Level Grammar (TLG - high-level formal specification language) IDE. TLG provide feature-rich environment for custom defined grammar, but we find that CDT has superior syntax support for C language as in [9]. Eclipse is a foundation for development of IDE Meta-tooling Platform [13] (IMP) that supports the creation of language-specific IDEs in Eclipse. In contrast to our solution, IMP gives very broad and general goal of developing IDEs. Our solution is targeting industrial proven IDE that is used for developing embedded applications. Each of these software packages promotes Eclipse as a highly adoptable working environment for development of IDE. That fact implies the purpose of this software solution, which is customized for multi-core platforms of the Cirrus Logic Company. Developed expertise is a solid basis for the future applications in this area.

Following sections cover the five main topics: Eclipse SDK, Target platform, CLIDE IDE, Testing and Conclusion. Eclipse SDK chapter describes purpose and basic Eclipse SDK functionalities. Target platform chapter contains basic information about target platform architecture and peripherals on this platform. CLIDE IDE chapter describes global architecture of CLIDE. This chapter also contains three subsections: Debug information handling, Instruction address representation in the source code editor and Setting and removing breakpoints. Debug information handling section contains UML class diagram of DWARF information [14] in CLIDE and description of the classes named for a debug information handling. Instruction address representation in the source code editor section contains UML diagram for drawing the addresses and description of appropriate classes. Setting and removing breakpoints section contain UML class diagram for breakpoint handling, as well as description of corresponding classes. Testing section describes CLIDE testing framework. Conclusion gives a short overview about the whole project.

2. ECLIPSE SDK

Eclipse development environment [1] is based on the IBM VisualAge [15] development environment. VisualAge development environment has been developed by the IBM since 2001 as an open source project. In 2003 project was transformed into the Eclipse Foundation. This foundation is a non-profit organization led by the IBM.

Eclipse SDK is a complete development environment written in the Java programming language. The main aim of the Eclipse is the development of applications in the Java programming language, with possibility of adding support for other programming languages via plug-ins. Also, Eclipse represents the environment where extensions are loaded, integrated, and executed. Eclipse SDK is used for development of a variety of a stand-alone applications as well as creating entirely new applications from existing components supporting the RCP (Rich Client Platform) concept [16].

Eclipse is a platform that serves as the development environment for the execution of other working environments for a variety of purposes that are not limited only to a programming, although that is the most usual case.

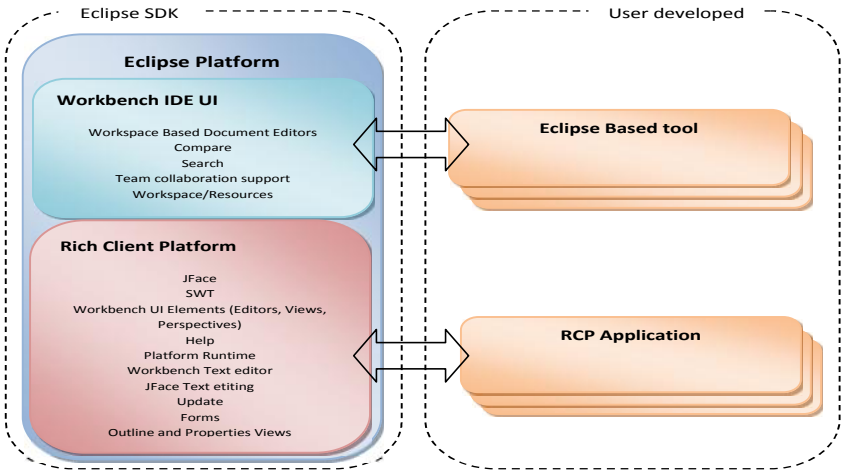


Figure 1 - Architecture of the Eclipse SDK

Figure 1. shows how the Eclipse SDK is built and how it can be expanded. Eclipse Platform can be divided into the two main parts: a part of the RCP and the working part - Workbench IDE. The main part of the RCP Runtime Platform is responsible for loading, launching and execution of all extensions. RCP uses SWT (Standard Widget Toolkit) graphics to build a graphical user interface subsystem. SWT elements are implemented as a set of graphical elements of the hosting operating system. JFace is Java application framework that is based on the SWT. It contains more complex elements built using the SWT's.

The entire graphical subsystem for communication with the user is built using the SWT and JFace. This graphical user interface is divided on the components (text editors, browsers, perspective, etc.) that can be reused in other extensions.

Workbench section is built on the Eclipse's RCP Runtime Platform. Workbench ensures uniform appearance of all windows and universal development process of all extensions by providing tools such as: search, teamwork, version control, unified system of help and a workspace for projects, files, etc.

3. TARGET PLATFORM

Developed IDE (CLIDE) is designed for the Cristal 32 digital signal processors family of the Cirrus Logic Company. This processors family contains a various processors that are differing in features and the amount of internal memory: CS495303 and CS-495313. CS4953xx integrates two 32-bit DSP processor which have separate memory area for data (X and Y), and instruction-specific memory zone of P. It has unified access to the X, Y and L memory zones, 64 words – bits wide. Beside two cores there is an adjustable DMA (Direct Memory Access) controller, which enables data transfer between peripherals, external memory or any internal kernel memory without CPU intervention.

Both cores are target-specific DSP processors with parallel architectures designed for fixed-point arithmetic with numbers in 2's complement representation. They are able to perform two independent memory accesses in a single cycle. They have eight 72-bit accumulator, and eight 32-bit registers, 4 X 4-data and Y-data register, and 12 index registers, which are used for memory access. Core structure can be seen in Figure 2.

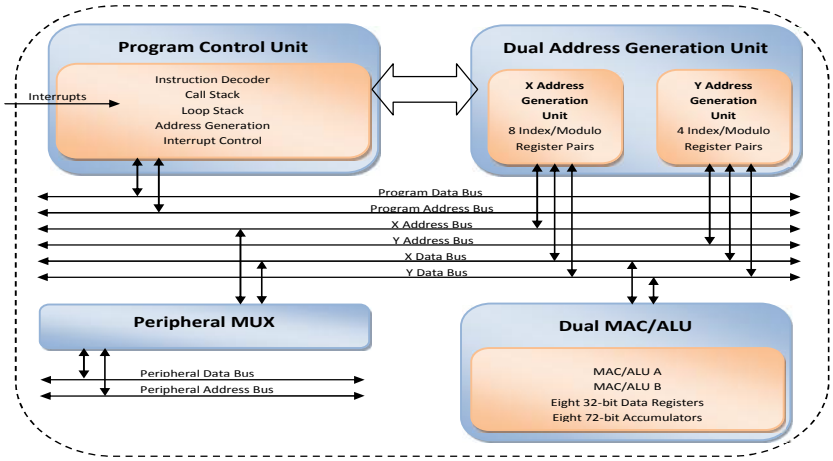


Figure 2 - Platform core structure

Peripherals that should be supported by the IDE are*:*

Digital Audio Input Port (DAI), Digital Audio Output Port (DAO), Serial Port Control (SCP), Parallel Control Port (PCP), External memory controllers, GP-IO, Inter Processor Communicator (IPC), Programmable Interrupt Controller, Timer, Global Configuration, Clock handler and Debug Controller (DBC)

4. CLIDE IDE

CLIDE is an Eclipse and RCP platform based IDE, developed for the programming support of the Cristal 32 DSP processors families. CLIDE could be extended to support different processor architecture with additional effort of adding new target model description and possible creation of editors for different assembly language syntax. This way heterogonous multi-core systems could be supported. CLIDE was developed using the Eclipse for the RCP Plug-in Developers. This package contains a complete set of tools for developers who want to create Eclipse plug-ins or Rich Client Applications. It includes a complete SDK, developer tools and source code, plus Mylyn, an XML editor and the Eclipse Communication Framework. RCP are standalone applications based on the components, with possibility of creating their own components. Beside that RCP applications could use finished Eclipse's components. CLIDE is implemented as an RCP application with the Eclipse's components used to build a new development environment, by adding new features in order to adjust IDE for the new target

platform. Figure 3. represents a basic Eclipse components that are used for CLIDE development, as well as components that are added by CLIDE to the Eclipse environment.

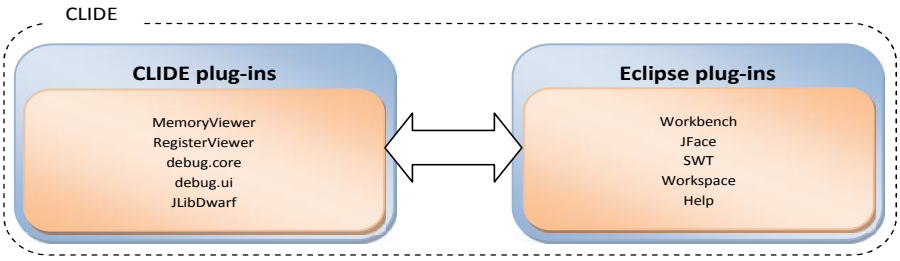


Figure 3 - CLIDE and ECLIPSE structure

Workbench is a set of components that form the basis of the graphical user interfaces (GUI). Workbench connects the GUI elements such as text editors, views, dialogs and other windows. Graphical user interface in the Eclipse environment is based on the SWT graphics library and JFace.

Workspace is an Eclipse component that combines active projects during the development time. Workspace handles source code files, projects and folders using the resources. Resources are a set of classes that represent projects, folders and files, which could be accessed through the three classes: IProject, IFolder and IFile. Resources represent the main set of classes that could be used for obtaining information about the files and projects in the Eclipse environment.

CLIDE's Component called MemoryViewer displays the contents of the Cristal 32 processors memory in the form of a table. This component provides the ability to display a part of a memory on provided address and memory area or assembly language symbols. Also, it is possible to modify the memory contents via this viewer.

RegisterViewer component provides the ability to view and modify the values of the Cristal 32 processors registers. Registers are organized in a tree structure divided into the groups. It is possible to choose the format of displayed register values.

Components debug.core and debug.ui are used to customize the functionality of the Eclipse's components for controlled program execution and debugging. JLibDWARF component is used to read debug information written in DWARF 2 format.

4.1. *Debug information handling*

Debug Information is closely related to the project and source code files [17]. All operations of the projects and files in the Eclipse environment are performed by the resources. Resources in the Eclipse allow adding files, folders, and projects. There are two types of these properties: session and persistent property. The difference between these two types of properties is found in fact that persistent property holds on between two Eclipse sessions, and a session property is lost after shutting down the Eclipse. CLIDE uses the session property that connects the project resources and files with classes that contain debug information. This approach avoids the search action of the list of projects and files in order to retrieve debug information. Figure 4. represent UML class diagram that handles debug information organization in CLIDE.

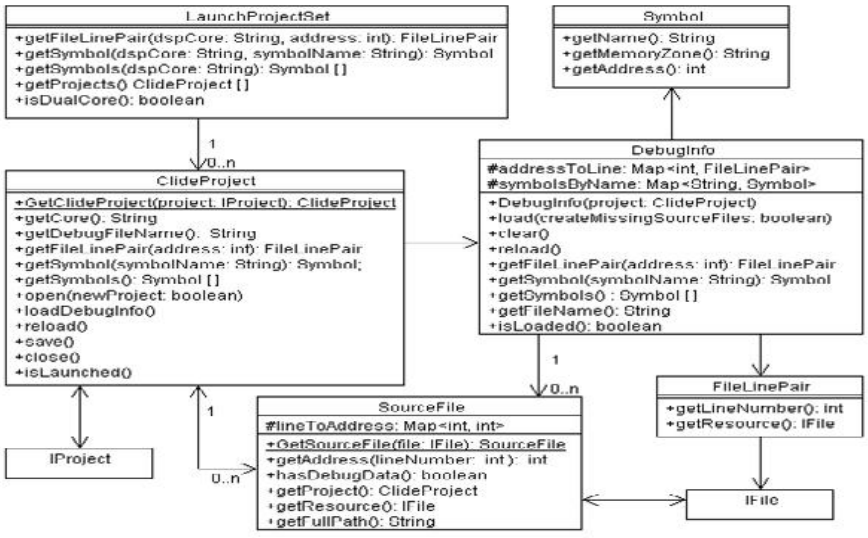


Figure 4 - UML diagram: DWARF information in CLIDE

The base class for debug information organization is the DebugInfo class. This class handles loading and organization of information. JLibDWARF library is in charge for loading information. Loaded information is organized in three groups:

- Information about symbols – This information is obtained from the DWARF tree.
- Mapped addresses to the line number and file name.
- Mapped line number to the file name and address.

First two groups of information are stored in the DebugInfo class, while third group is distributed through the SourceFile class instances. This division is based on the principle that the SourceFile class handles requests which obtain address from a file and a line number, and the DebugInfo class serves all other requests. SourceFile class links file resources with debug information related to that file. File resource is linked with SourceFile class through session property mechanism. This type of linking provides obtaining SourceFile class instances from a resource, without searching through the projects. SourceFile class is starting point for access to information based on a file and a line number. CLIDEProject class is linked with project resource (IProject) through session property mechanism. This class integrates debug information related to one project.

CLIDE supports simultaneous project execution. These projects are interconnected, executed on the same processor or simulator and cooperative. Simultaneous launching of multiple projects implies usage of several sets of debug information. LaunchProjectSet class handles such scenario. This class integrates information of launched projects. LaunchProjectSet class is a starting point for access to information based on the address and information about symbols.

4.2. Instruction address representation in the source code editor

Low level of abstraction in assembly language obstructs developers overall view, so IDE should provide easy access to instructions addresses in program memory and data addresses in data memory. Information about instructions addresses in CLIDE are represented in the column on the left side of the editor.

This column shows appropriate address beside every source code line which contains instruction. Eclipse provides text editors the feature for adding these lines through texteditor.rulerColumns access point.

CLIDE joins AddressBar class to this access point. AddressBar class extends AbstractRulerColumn class which defines functionality for drawing the columns. Figure 5. shows UML class diagram of classes involved in drawing the column with address.

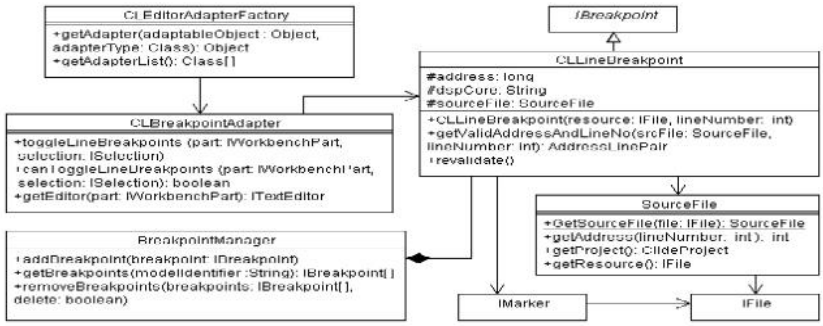


Figure 5 - Class diagram for drawing the addresses

AddressBar class takes the resource from the CLEditor class that corresponds to the IFile. Static method getSourceFile() takes the SourceFile object. This method use session property mechanism for linking the SourceFile and the IFile classes. The SourceFile class takes the address based on the line that should be drawn.

4.3. Setting and removing breakpoints

Eclipse supports extension of breakpoint functionality by adding new types of breakpoints. CLIDE adds its own breakpoint type through CLLineBreakpoint class. Setting and removing breakpoints could be done only in the source code editor. Eclipse provides org.Eclipse.core.runtime.adapters extension point for adding standard source code editor actions. CLIDE implements that plug-in through the CLEditorAdapterFactory class. This class adds adapter for setting and removing breakpoints. Adapter is implemented with the CLBreakpointAdapter class.

Adapter sets breakpoint by creating a new instance of the CLLineBreakpoint class and adding it to the record in the BreakpointManager class which represent breakpoint handler inside the Eclipse environment. Breakpoints are removed by deleting them from the record of breakpoint handler. Figure 6. shows the UML class diagram of breakpoints handlers.

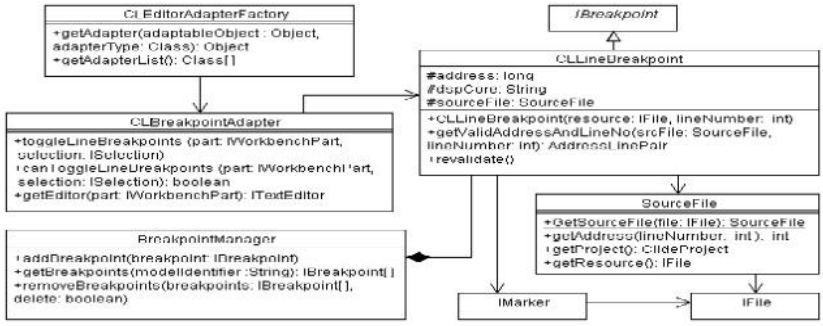


Figure 6 - UML class diagram for breakpoints handling

CLLineBreakpoint represent breakpoints in CLIDE. This class extends the IBreakpoint abstract class in order to integrate with the Eclipse environment. To each CLLineBreakpoint class object is associated one marker. Markers in Eclipse are used for marking source code snippets in the files. There are various types of markers: breakpoints, errors, warnings, etc. For description of breakpoints in files, CLIDE use CL.markerType.lineBreakpoint type of marker.

BreakpointManager class handles all breakpoints in the Eclipse environment. This class takes the record about all breakpoints in the system. Setting a breakpoint requires obtaining address in compiled program that corresponds to the source code line where breakpoint wants to be set. Required address is obtained using the SourceFile class. This class contains debug information about link between source code line and address.

Appropriate SourceFile object could be obtained based on the file resource using the static method getSourceFile(). Then, getAddress() method is used for obtaining required address. Same information is used for validation of breakpoints. After loading of debug information it is possible that some of breakpoints are on the source code lines that don't have corresponding address in compiled program. These types of breakpoints are moved to the next valid address or if that is not possible, they are removed from the record.

5. TESTING

Reliable final product depends largely on the testing framework. CLIDE's testing framework had a proportional grow of the number of test cases as new features arrived. That was necessary, because final goal was covered testing of the most of the CLIDE's functionalities. CLIDE's Testing framework was organized in JUnit testing format. Two major groups of tests were recognized:

- UI tests – Testing of UI functionalities,
- Model tests – Testing of non UI functionalities.

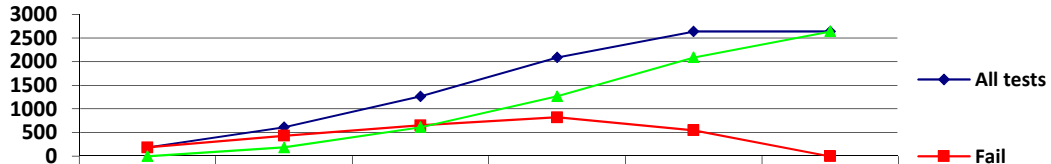
UI tests validate UI functionalities through the execution of well defined activities. For example, when action "Create new Project" is launched than it should be followed by appearance of the "New Project" wizard, and if that is the case test will pass, otherwise it will fail.

Model tests are designed to test non UI functionalities. For example, after successful "Create new Project" action execution, result of that should be the new project in the workspace with valid properties. If that is the case test will pass, otherwise it will fail.

These two distinguished groups of tests were designed to cover validation of CLIDE's properties used for interaction with users as well as various CLIDE's actions used for project development and debugging.

Most of the time Test-Driven development technique was accepted as the mainstream of the development process. This technique implies write tests first than develop. Initially there were only 132 UI tests and 52 model tests and all were failing because functionalities weren't implemented yet, just specified. Development process was repeated until all tests were passing. When that point was reached, then, new modules were scheduled for development process and, as described previously, writing JUnit tests, by using the task specification, represent a starting point in that process. Table 1. represents the data flow of testing results thru the milestones. Aim of all milestones was to make all tests from previous milestone pass and to define new task thru the new test cases. This was not applied only for the first and the last milestones. Purpose of the first milestone was to define starting point of development process thru 184 JUnit tests, and the last milestone aim was to make all tests pass. Currently 2641 JUnit tests forms CLIDE's testing environment. All these tests are launched on CLIDE's source code update. That way regression testing is established and developers have much more confidante to change existing code. This is not a final number of JUnit tests and it is increased every time when the new feature requests arrive.

Table 1. Testing results



	1st Milestone	2nd Milostone	3rd Milestone	4th Milestone	5th Milestone	6th Milestone
All tests	184	616	1269	2092	2641	2641
Fail	184	432	653	823	549	0
Pass	0	184	616	1269	2092	2641

6. CONCLUSION

CLIDE represent an industrial proven IDE that is used for development of embedded applications. It was estimated that CLIDE roughly doubled developer productivity. This estimation was based on a man/month assessment for referenced project in a pre-CLIDE and CLIDE period of time. Most significant acceleration of the development process is established in the debugging area. This is due to many useful properties that CLIDE offers, such as: controlled program execution, expression view, memory view, etc. CLIDE, also, gives a developer comprehensive picture of programming model that could be used in order to achieve more efficient source code and hence a higher quality of a developed software.

Usage of the Eclipse and RCP framework as a foundation were right choice to make. Extensible nature of the Eclipse and RCP framework was the main feature that saved time in CLIDE development.

Finally, easy implementation of the JUnit test leaded to development of the 2641 JUnit tests that validate most of the CLIDE's functionality, thereby guarantee reliable working IDE.

7. ACKNOWLEDGMENT

This work was partially supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia under Grant TR-32031

REFERENCES

[1] 2014, July) Eclipse. [Online]. http://www.eclipse.org/

[2] J., Wiegand, J. Des Rivieres, "Eclipse: A platform for integrating development tools," IBM Systems Journal, vol. 43, no. 2, pp. 371 – 383, 2004.

[3] (2014, July) Texas Instruments - Embedded Processing & DSP. [Online]. http://www.ti.com/lit/-sg/sprt285e/sprt285e.pdf

[4] (2014, July) Analog Devices – Processors and DSP. [Online]. http://www.analog.com/en/processors-dsp-/products/index.html

[5] (2014, July) Silicon Labs. [Online]. http://www.-silabs.com/Pages/default.aspx

[6] (2014, July) Freescale. [Online]. http://www.freescale.com/

[7] (2014, July) Renesas. [Online]. http://www.renesas.com/products/index.jsp?campaign=gn_prod

[8] (2014, July) Mentor Graphics. [Online]. http://-www.mentor.com/embedded-software

[9] Matti K., Mikko K., Pekka R., Tommi M. Juha J., 2006, "Developing an Open Source Integrated Development Environment for a Mobile Device," in ICSEA '06 Proceedings of the International Conference on Software Engineering Advances, Washington, DC, p. 55.

[10] Anders P., Dag N., Thomas N., Paul P., Ivica C. Severine S., 2009, "Save-IDE: An Integrated Development Environment for Building Predictable Component-Based Embedded Systems," in ASE '08 Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, Washington, DC, pp. 607-610.

[11] Leif Frenzel, 2007, "Experience report: building an eclipse-based IDE for Haskell," in ICFP '07 Proceedings of the 12th ACM SIGPLAN international conference on Functional programming, New York, NY, pp. 220-222.

[12] Xiaoqing W., Fei C., Shih-hsi L., Wei Z., Chunmin Y., Barrett R. B., Jeffrey G. G. Beum-Seuk L., 2003, "T-Clipse: an integrated development environment for two-level grammar," in eclipse '03 Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange, New York, NY, pp. 89-93.

[13] Robert M. F., Stanley M. S. Philippe C., 2007, "IMP: a meta-tooling platform for creating language-specific ides in eclipse," in ASE '07 Proceedings of the twenty-second IEEE/ACM international conference

on Automated software engineering, New York, NY, pp. 485-488.

[14] (2014, July) DWARF Debugging Information Format. [Online]. http://www.dwarfstd.org/doc/dwarf-_1_1_0.pdf

[15]L.A., Lymer, S.F., Ryman, A.G. Chamberland, 1998, "IBM VisualAge for Java," IBM Systems Journal, vol. 37, no. 3, pp. 386 – 408

[16]A., Reiswich, E. Kornstadt, 2010. "Composing Systems with Eclipse Rich Client Platform Plug-Ins," Software, IEEE, vol. 27, no. 6, pp. 78 – 81, Nov. Dec.

[17][17] (2014, July) How to write an Eclipse debugger. [Online]. https://www.eclipse.org/articles/Article-Debugger/how-to.html

SUMMARY

INTEGRISANO RAZVOJNO OKRUŽENJE ZA VIŠE JEZGARNE SISTEME

Razvoj programskih rešenja koja pružaju komforno radno okruženje za razvoj programske podrške za platforme sa ograničenim resursima je oduvek predstavljalo izazov. Da bi se navedeni cilj postigao bilo je neophodno integrisati sve specifične alate razvijene za tu namenu. Ovaj rad opisuje Integrisano Razvojno Okruženje koje je dizajnirano kako bi zadovoljilo sve specifičnosti razvoja programske podrške za familiju više jezgarnih platformi namenjenih digitalnoj obradi signala kompanije Cirrus Logic. Kao osnova za razvoj korišćena je Eklips (Eclipse) platforma i RCP (Rich Client Platform), zato što pruža proširiv i prilagodljiv sistem za razvoj integrisanih razvojnih okruženja. Realizovano je CLIDE (Cirrus Logic Integrated Development Environment), kao pouzdano razvojno okruženje za razvoj programske podrške. Validacija rešenja je ostvarena kroz 2641 JUnit ispitnih slučajeva koji ispituju većinu funkcionalnosti koju pruža CLIDE. CLIDE u značajnoj meri povećava kvalitet izrade programske podrške namenjene više jezgarnim sistemima i smanjuje potrebno vreme razvoja programske podrške, čime opravdava troškove razvoja integrisanog okruženja.

Ključne reči: *integrisano razvojno okruženje, više jezgarni sistemi, Eklips*