Debugging Reflex-programs on digital plant models

Alexandr Dvinianin
Institute of Automation and Electrometry
Siberian Branch of the RAS
Novosibirsk, Russian Federation
a.dvinianin@g.nsu.ru

Tatiana Liakh
Institute of Automation and Electrometry
Siberian Branch of the RAS
Novosibirsk, Russian Federation
antsys_nsu@ mail.ru

***SAbstract*—Reflex is a process-oriented language designed for the development of control software in cyber-physical systems. Cost of errors in such systems is very high. Dynamic debugging allows to check the quality of control software and prevent breakdowns. However, there are no debugging tools for the Reflex language. In this paper, we describe a dynamic debugging module for the Reflex language. The debug module can be used with AVR168/328 microcontrollers as well as on the user's PC. Input port values can be managed either manually or by a simulated control plant, also described in the Reflex language. Users can debug Reflex program in step-by-step and cycle-by-cycle mode.**

Keywords—Process-oriented programming, Dynamic debugging, Digital plant, Software simulation, Cyber-physical systems, Control software.

I. INTRODUCTION

The development of software for cyber-physical systems (CPS) is associated with a number of difficulties. CPS software has a number of features which complicates development:

- Openness - software interacts with an "open world" through sensors and gates
- Event-driven behaviour - software generates a control signal according to the data received from the external environment
- Logical parallelism - control algorithm consists of many parallel independent or loosely coupled processes
- System operation time is not limited - software must be functional until a special command is received
- Synchronism - an external environment is an inert physical object that does not change its state instantly

This leads to the fact that CPS software developers use specialized language tools (such as IEC-61131[1], G LabVIEW[2], etc.). One of such language tools is Reflex[3,4], a process-oriented language developed by the Institute of Automation and Electrometry and intended for programming logic control devices. A Reflex program is a complex of processes. Each process is a state machine with a set of dedicated states and a time service.

The Institute of Automation and Electrometry SB RAS is developing an integrated development environment for the Reflex language called RIDE. RIDE assumes the basic functionality of development environments, static and dynamic verification on a virtual control plant, as well as the possibility of dynamic debugging.

Due to the high cost of errors in the development of CPS software, dynamic debugging is one of the most common ways to control software quality. For the Reflex language, a translator to c-code and code for Atmega16 have been developed, but there are no built-in dynamic debugging tools in the environment.

The article presents debugging technology for Reflex programs on Atmega microcontrollers using digital models of a control plant.

Section II of the article describes various debugging technologies used for microcontrollers. Section III describes the developed debugging approach and architecture of the software package. Section IV provides further plans.

II. DOMAIN ANALYSIS

In the field of microcontrollers, various interfaces are used for debugging. The most common are JTAG[5] and SWD[7] interfaces.

The JTAG interface was developed for testing chips and boards[6], but later it became possible to flash and debug. SWD is an ARM special protocol designed specifically for on-chip debugging. Both protocols allow you to debug the program step by step or using breakpoints and also allow read/write operations of memory, but unfortunately, to work with these interfaces, expensive additional equipment is required.

Also, the CODESYS[8] software package and debugWire[9] and debugMon[10] interfaces were analyzed. CODESYS is intended for the development of programs for programmable logic controllers, allow step-by-step and breakpoint debugging, and read/write operations.

DebugWire and debugMon interfaces were developed for debugging microcontrollers with limited resources, such as 8-bit AVRs.

In addition to the above, there are many handwritten libraries, most of which allow information to be output through the UART serial interface.

Thus, the standard set of functions of a modern debugger is step-by-step and breakpoint debug modes, memory read/write, and the ability to emulate a microcontroller.

III. Reflex-code debugging technology

The developed debugging algorithm is shown in Figure 1. The algorithm starts with initialization (red area), in which the debugger receives information about processes and variables from configuration files that were automatically generated by the translator from the Reflex code (1), and also receives physical addresses of variables from a microcontroller (2) and passes the "next" command (3). Then comes debugging by itself (blue area): the debugger waits for information about the state, and the microcontroller executes the Reflex code before calling the debug (4) function, then it transfers its state to the debugger (5) and waits for the next command (6). The debugger processes the received state (7) and sends the "next" command (8), after which the algorithm goes to step 4.

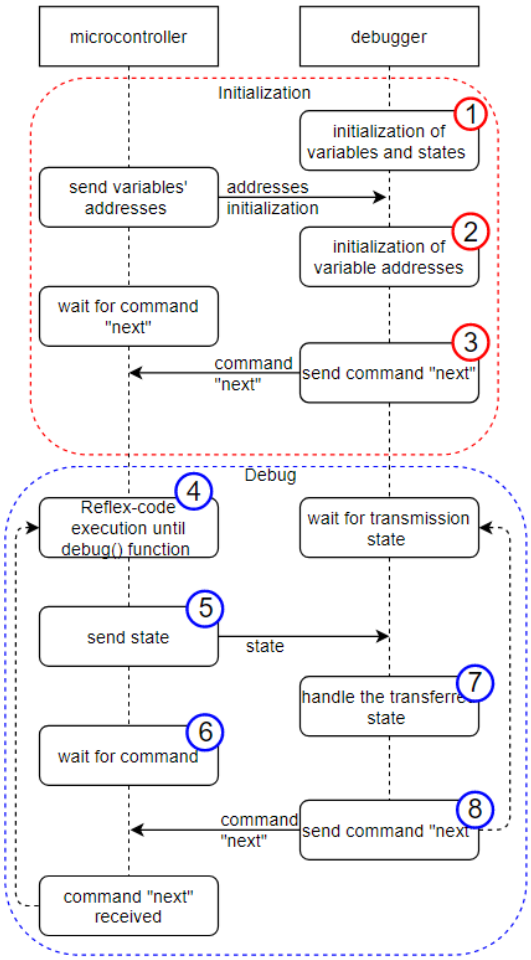

Fig. 1. Debugging algorithm

If it is necessary to change the value of a variable (Fig. 2), the debugger sends a writing packet consisting of the size of the variable, its address and the value to be written. The microcontroller executes the command and continues to wait for the "next" command. Figure 3 demonstrates packet format of initialization, state transfer, and setting.

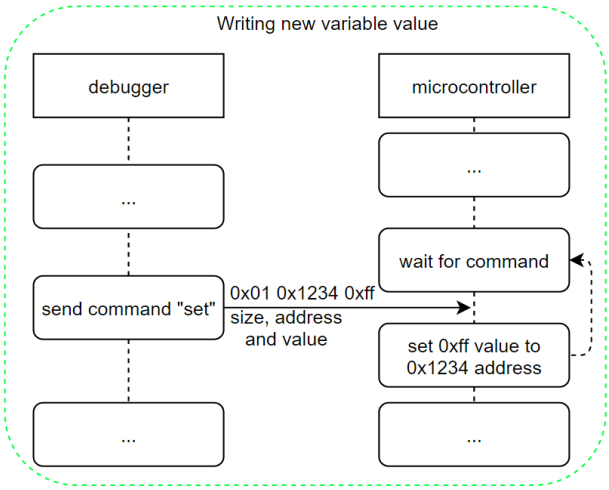

Fig. 2. "Write" command algorithm

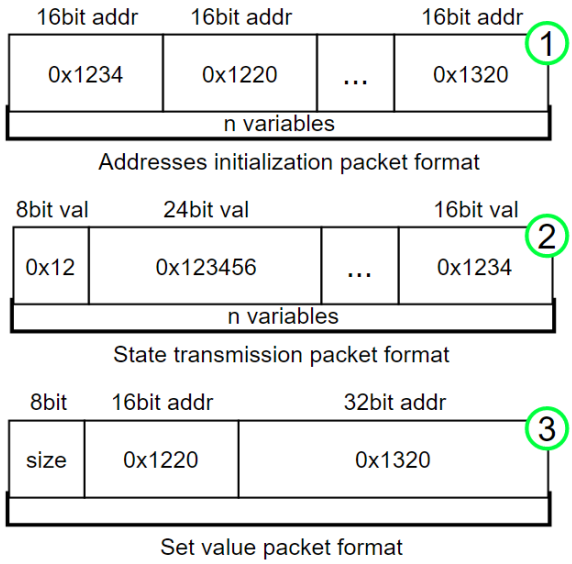

Fig. 3. Initialization (1), state transfer (2) and write (3) packet format

IV. Architecture

The debugger architecture is shown in Fig. 4. The modified translator of Reflex-code (2) according to the rcs source file (1) generates source codes for compilers (3, 9), as well as files with information about variables and processes (7) for the debugger (8). Further, the code for the microcontroller is compiled (10) and flashed (11), and an executable file is compiled (4) from the c-code source, which the debugger program can run (5). Next, the debugger interacts via a virtual COM port with a microcontroller (12), or, if necessary, with a plant simulator, also developed in Reflex (6).

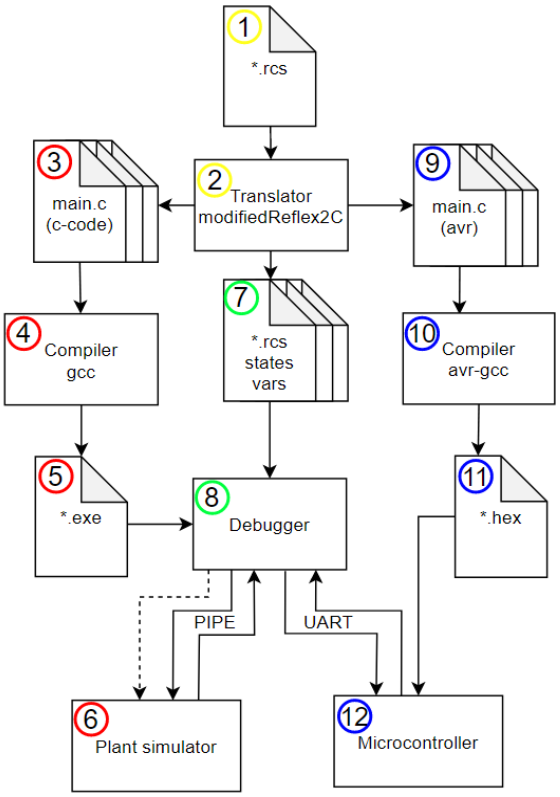

Fig. 4. Architecture of debugger

V. Interface

On launch user chooses the name of the project (Fig. 5, 1), as well as whether to connect to a virtual port or run a simulated process (2). It is also possible to choose a variable name and send a command to write (3). The user can choose one of the debugging modes: step-by-step, tact-by-tact or breakpoints.

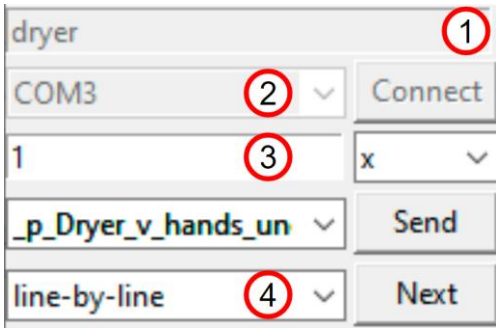

Fig. 5. Control panel

The window on Figure 6 displays processes of program (1), the current state of the process (2) and state setting time (3).

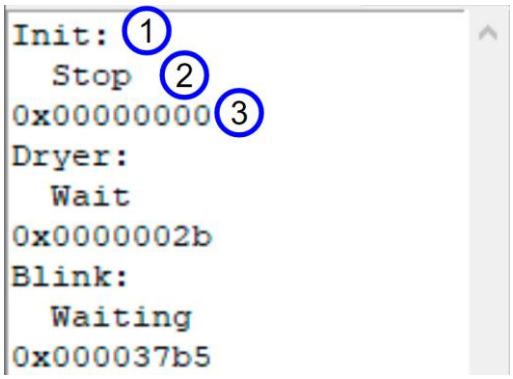

Fig. 6. Window with state of processes

Figure 7 displays a window with state of all variables (1) and a window with Reflex code (2). The current executable line is highlighted in green (3).

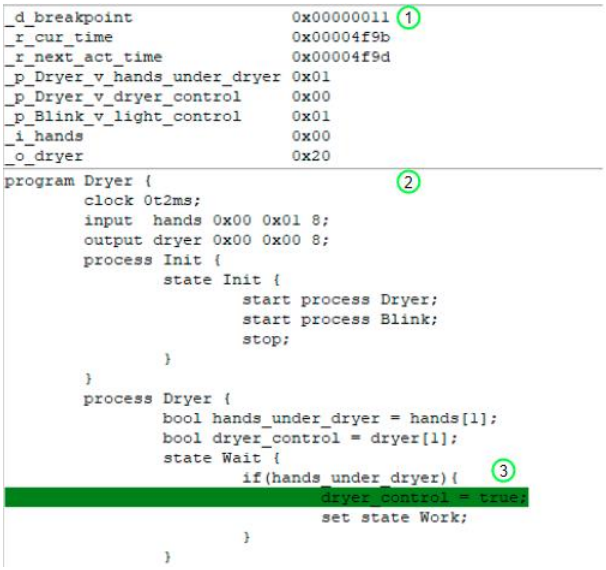

Fig. 7. State of variables (1) and Reflex-code (2)

VI. CONCLUSION

As a result of the work, a software complex for debugging Reflex programs was developed. The developed technology makes it possible to debug the Reflex code on the AVR168 microcontroller, including:

- Step by step debugging
- Debugging with breakpoints
- Matching Reflex Code Strings
- Reflex code execution on the simulator
- Manual debugging and debugging on a virtual control object

In the future, we plan to integrate the debugging module into web-IDE RIDE for process-oriented languages.

VII. References

[1] IEC 61131-3: Programmable controllers. Part 3: Programming languages. 2nd edn. International Electrotechnical Commission. (2003)

[2] Travis J., Kring J. LabVIEW for everyone: graphical programming made easy and fun. 3rd edn. Prentice Hall PTR, Upper Saddle River, NJ, USA, (2006)

[3] Liakh T.V., Rozov A.S., Zyubin V.E.: Reflex language: a practical notation for cyber-physical systems. System Informatics. 12, 85–104 (2018)

[4] Anureev I.S: Operational semantics of Reflex. System Informatics. 14, 1–10 (2019)

[5] 1149.1-1990—IEEE Standard test access port and boundary-scan architecture. Institute of Electrical and Electronics Engineers (1990)

[6] Lau S.: Reinventing JTAG for SoC debugging, embedded.com/reinventing-jtag-for-soc-debugging, last accessed 2021/04/10

[7] Williams M.: Low pin-count debug interfaces for multi-device system, semanticscholar.org/paper/Low-Pin-count-Debug-Interfaces-for-Multi-device-Williams/12d9b9320b74dfd0d00123f5d271e9ca65c9128b, last accessed 2021/04/10

[8] Hanssen H.: Programmable logic controllers: a practical approach to IEC 61131-3 using CODESYS. John Wiley & Sons Limited, the Atrium, Southern Gate, Chichester, England, W Sussex, Po19 8sq (2015)

[9] The debugWire protocol, ruemohr.org/docs/debugwire.html, last accessed 2021/04/10

[10] CortexM3 Technical Reference Manual, developer.arm.com/documentation/ddi0337, last accessed 2021/04/10