PROBLEMY MECHATRONIKI
UZBROJENIE, LOTNICTWO, INŻYNIERIA BEZPIECZEŃSTWA

ISSN 2081-5891

11, 2 (40), 2020, 95-110

PROBLEMS OF MECHATRONICS
ARMAMENT, AVIATION, SAFETY ENGINEERING

Fire Control System Software for Remote Controlled Weapon Stations: History, State of the Art and Opportunities for Future Development

Rafał KRUSZYNA

Zakłady Mechaniczne Tarnów S.A.
30 Kochanowskiego Str., 33-100 Tarnów, Poland
Author's e-mail address and ORCID:
rafal.kruszyna@gmail.com; https://orcid.org/0000-0003-2110-7758

Received by the editorial staff on 13 July 2018
The reviewed and verified version was received on 08 June 2020

DOI 10.5604/01.3001.0014.1996

Abstract. This paper presents the evolution of fire control system software for Remote Controlled Weapon Stations (ZSMU in Polish) developed and manufactured by Zakłady Mechaniczne "Tarnów" S.A. (Poland). The paper describes the architecture, scope and purpose of research and development of the fire control system software, and the prospective directions of its future development. A complement to the paper is a specification of the optimisation methods used during the development of the fire control system software and the work organisation of a team of programmers working on the source code of the fire control system software. The paper illustrates how the consistent execution of research and development improves the effectiveness, scalability, and optimisation of the ZSMU fire control system software.
Keywords: software engineering, fire control system, software

This work has been compiled from the paper presented during the 12th International Armament Conference on Scientific Aspects of Armament & Safety Technology, Jachranka, Poland, September 17-20, 2018.

1. INTRODUCTION

Zakłady Mechaniczne "Tarnów" S.A. (ZMT S.A., Poland) has been manufacturing Remote Controlled Weapon Stations (ZSMU in Polish) for many years. The first ZSMUs were designed at ZMT S.A. in cooperation with third-party contractors. The design engineering team at ZMT S.A. was tasked with the mechanical design and weapons engineering; the third-party contractors designed the electronic solution and software. As ZMT S.A. grew, it generated capital expenditure to develop its R&D business, the effect of which was the expansion of the Electronic Design Engineering business and the establishment of a new software development unit. In 2014, ZMT S.A. launched a project to develop a remote controlled weapon station, version code ZSMU-1276 A4. A feature of the ZSMU A4 module was that its electronic solutions and software are completely proprietary designs of the ZMT S.A. Design Engineering team. ZMT S.A. acquired new competencies and project experience. This paper describes and discusses the development process of the main software solution for the control PC of a whole family of ZSMUs between the years 2014 and 2018.

2. FOREDESIGN

Certain assumptions had to be defined to enable the development of the ZSMU system software. The assumptions determined the final architecture and other features of the system. The desired features of the final system software were:

- Full operation of the capabilities of the hardware the system would be implemented in;
- Low development costs;
- Operating stability and freedom from errors;
- Short development time;
- Maximum responsiveness;
- Flexible expandability.

The portability of the system was to be maximised to enable easy implementation in newer products of the same class. All those features would ensure that the system software needed not to be written from scratch for every new ZSMU designed with functionalities like its predecessors. It was not possible to satisfy all these conditions; they contradicted each other more than once.

For example, there was a rudimentary decision to be made to choose the programming language in which to write the system. Let us assume that the programming languages to choose from included an assembler, C, C++, and Java. Each of these programming languages meets diverse requirements regarding the system.

The assembler allows writing applications which use specific processor commands to achieve the maximum performance possible. The assembler is useless in terms of code portability, flexibility, code transparency, and the logical architecture of the application to be written with it. C is a hardware-independent programming language. The code written in C performs worse than the code from an assembler; however, the performance is relatively high because of the nature of C, which was designed as a programming language for the execution of low-level operations [1]. C is encumbered with many drawbacks: it has only one namespace, which restricts scalability to a certain degree, and no automatic memory management is provided, which can easily result in memory management errors. In terms of the logic of an application's architecture, the lack of any object-oriented programming imposes a serious constraint — especially in large projects which require a high degree of code transparency. C++ is yet another programming language to evaluate. In terms of performance, C++ code is compiled to native instructions set, providing a high-performance final application [2].

By applying appropriate techniques, the finished application performance can be like that of applications developed in C. A potential drawback is that the programming language has no automatic memory management. This forces programmers to write applications specifically with memory management in mind; under specific circumstances, it might result in many errors and operating instability. An undisputed advantage is that C++ is an object-oriented programming language. It significantly improves the transparency of code and system architecture in complex applications.

The namespaces also help develop projects with high scalability. A rather extensive standard library (and the availability of many other libraries) is another advantage of C++. Java is also a programming language of consideration. It is an extremely popular language and provides programmers with multiple convenience features. Java was specifically built to make programming easier. A programmer writing in Java focuses on the problem to be solved and not the complexity of the programming language. Java is cross-platform, features automatic memory management and fully supports OOL. While it seems to be the perfect solution, these convenient features come with their own price tag. Java requires a virtual machine to run, which is a considerable processing overhead; the computing load is further increased by the memory management automation features. Despite its advantages, Java is ultimately much worse in performance than C or C++. The poor performance alone disqualifies Java from real-time control applications.

3. APPLIED TECHNOLOGIES

Having analysed the specification requirements for the final system, the level of system complexity and a few other applicable parameters, preliminary project decisions were made to define the IT technologies which should be applied. The project management decided that the fire control system (FCS) software would be written in C++ with a Qt library support. The hardware platform chosen for the project was an x86 military-grade PC managed by a Linux kernel-based operating system.

4. ARCHITECTURE

4.1. Hardware architecture

To facilitate the understanding of the architecture of the FSC software developed for the ZSMUs manufactured by ZMT S.A., a pictorial diagram is provided below.

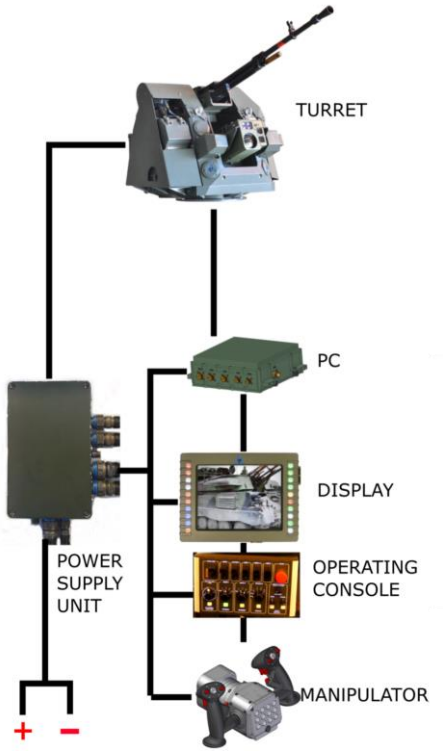

Fig. 1. ZSMU A4 turret system

Figure 1 illustrates an overview of the hardware structure for the ZSMU. Depending on the mission specifications, the hardware structure can be expanded with more operating stations or connectivity with external systems. Depending on their models, the ZSMU manufactured at ZMT S.A. vary in hardware structure, although they feature many common components. Each ZSMU has a power supply unit (controller), a display of the video feed transmitted from the electronic optical cluster and operation of function keys, an operating console as an input interface of the FCS settings, a joystick, a control PC, drive controllers, a weapons controller, and the electronic optical cluster. Figure 2 illustrates a detailed block diagram of the most extensive ZSMU version, the ZSMU A5.

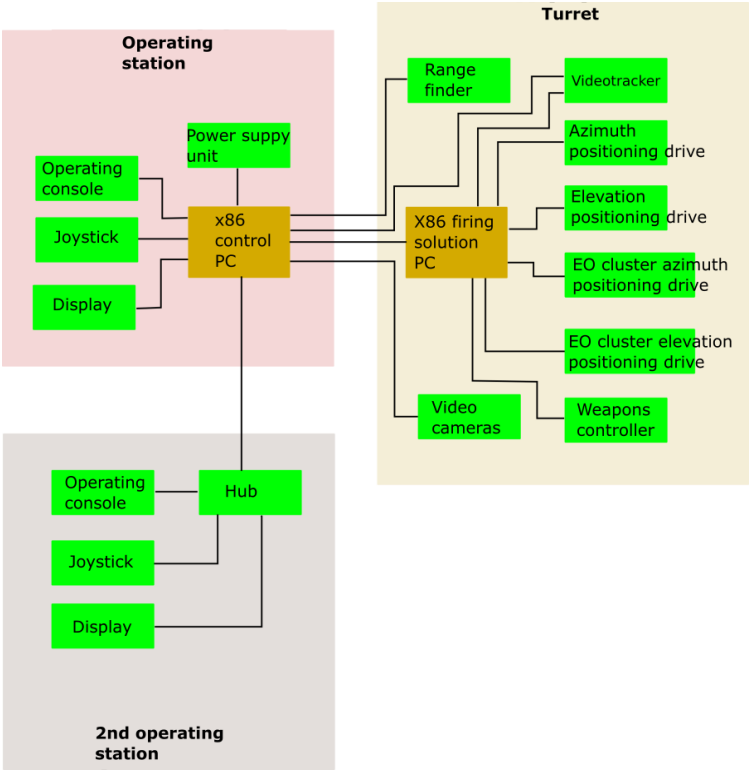

Fig. 2. Block diagram of the ZSMU A5

Colour brown designates the control computers. Green designates the physical hardware which the computers communicate with and control. The peripheral varies with the ZSMU turret version. Example: the ZSMU turret versions – A4, A3, A5, and A3B – have different operating consoles, joysticks, weapons controllers, electronic optical clusters, and hubs.

4.2. System architecture

One of the greatest challenges in the development of the ZSMU FCS software was to build a suitable system architecture. Two strategies of system architecture development were considered. The easy one consisted in writing a separate software solution for each tower version. The approach was straightforward, a code would be written for each ZSMU model so that its FCS software would just cover the hardware functionalities. A good solution in the short term, it would cause many problems in the long-term evolution of the product versions. The first of the main drawbacks was that writing a separate FCS software solution for each ZSMU version would be much more labour-intensive in the long term than developing a unified and universal platform. Between 2014 and 2018, ZMT S.A. developed four new ZSMUs; developing the FCS software for each of the four versions would require more time and resources than a single, universal FCS software solution. The second undisputed advantage of developing single, universal FCS software solution was that its debugging, optimisation, and patching would be automatically applied to all supported ZSMUs. It was the only reasonable choice to develop a unified universal FCS software platform.

Another considerable challenge for the FCS software development was the variety of hardware in the ZSMUs. Each ZSMU varies in the applied mechanical and electronic solutions. A question has arisen: how to unify this variety to achieve the maximum possible performance with a code transparent enough to enable easy analysis and future evolution of the FCS software? An example of the problems at this stage was that certain ZSMU versions (A3 and A3B) featured one control PC each, while other ZSMU versions (A4 and A5) had two control PCs each: one at the operating station and the other in the ZSMU artillery tower assembly. The challenge in support between one and two control PCs was solved by dividing the FCS software into two main processes. One process manages the graphical user interface and operating controls. The other process manages drives, controllers, stabilisation, and ballistics. The twin-process division of the FCS software solved the conflicts between the single and twin control PC systems. A single control PC military turret has both processes launched on the same PC unit. The processes exchange data over IPC. A twin control PC military turret has the GUI launched on one of the control PCs. The drive control process is launched on the other control PC. Both processes exchange data over a physical link.

From the system architecture perspective, there is no difference between the single and the twin control PC configuration. A simplified UML diagram of the architecture of both processes is shown in the two following diagrams.

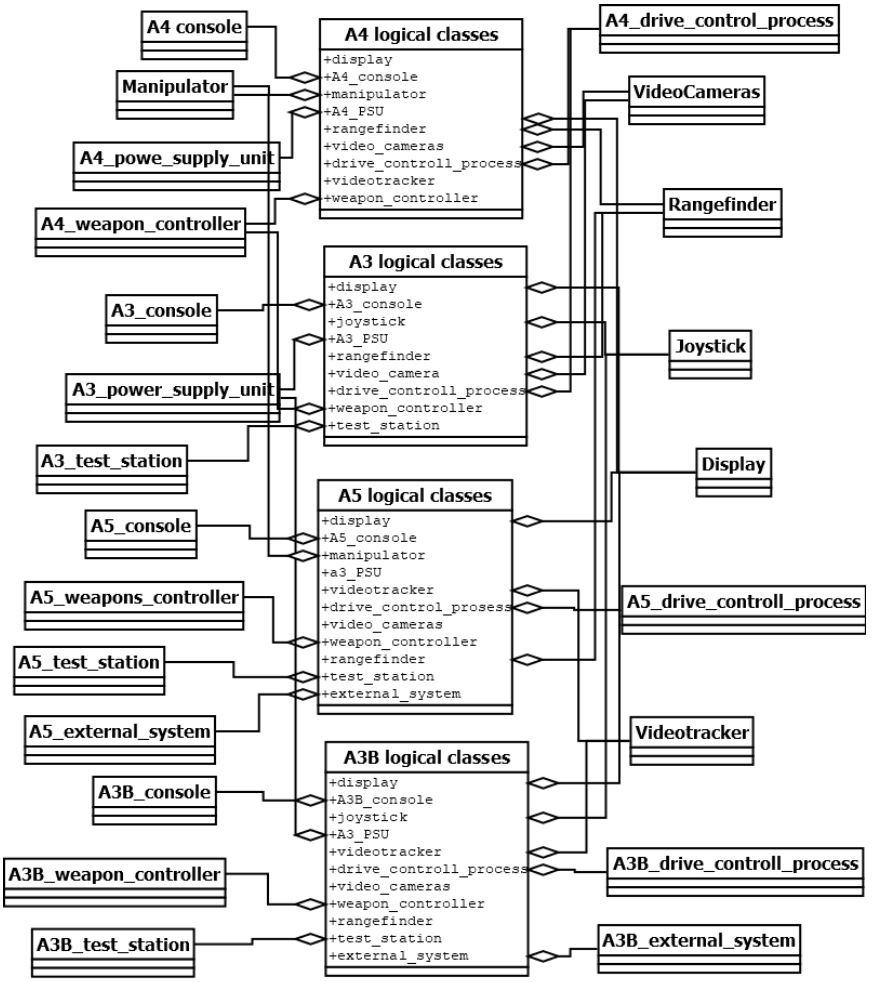

Fig. 3. Overview of the GUI process architecture

Figures 3 and 4 provide simplified illustrations of the concept behind the FCS software architecture for the ZSMU military turrets designed by ZMT S.A. The actual arrangement of the classes is much more complex. The diagrams merely illustrate the idea behind the applied solution. The diagrams show the solution to the hardware diversity between the specific configurations of ZSMUs.

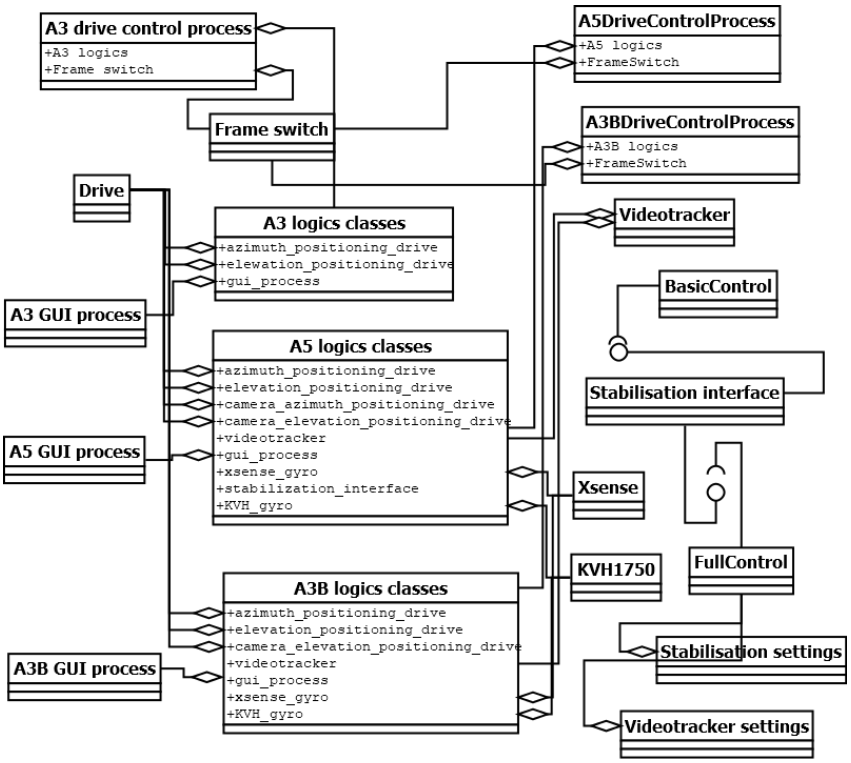

Fig. 4. Overview of the drive control process architecture

The problem was solved by strict cooperation with the electronic engineering team. The electronic solution design required the hardware units of the ZSMU military turret to be autonomous and independent from one another. This electronic solution design provided the PSUs, operating consoles, displays, joysticks, weapons controllers, videotrackers and other hardware units fully autonomous, and the operation of every hardware unit was not correlated to other hardware units in any way. The only correlation between any two hardware units is software-driven with an unlimited degree of flexibility. The hardware support is a set of independent and separate classes. For each hardware unit in the FCS, a code portion was written forming a hardware driver independent from other hardware drivers. The Figures 5 and 6 in the UML notation present two examples of the classes; one supports a drive controller, and the other supports the joystick.

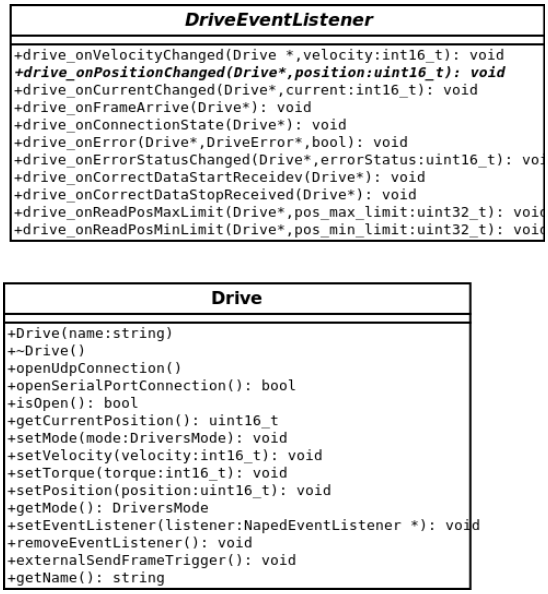

Fig. 5. Drive controller

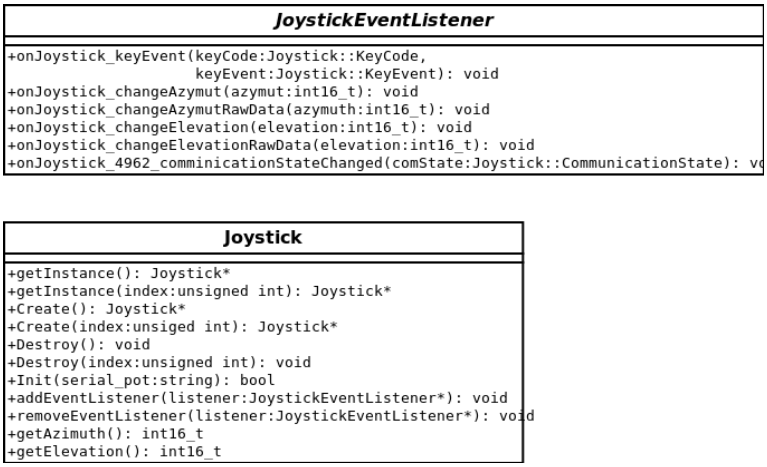

Fig. 6. Joystick controller

The modular design of the drivers makes the FCS a very flexible solution which permits a relatively quick implementation of new functionalities and fast porting of the FCS to new products. A major aspect consists in the separation of the control logic between the individual models of the ZSMU remote controlled weapon stations. The idea is clearly illustrated in Figures 3 and 4.

Every new ZSMU model receives dedicated classes which handle the logic mapping between the hardware units of the ZSMU. As a result, changes in the logic of the operation of an ZSMU do not result in changes in another ZSMU; this provides full flexibility of functional configuration for each ZSMU military turret version. The implementation of a new ZSMU is also a relatively easy task. This action requires writing its logical classes. This is followed by using the already written drivers applied in the existing ZSMUs — which are compatible with the new ZSMU – and mapping the drivers to the logical classes. If a driver is missing (due to non-compatibility with the existing ZSMU models), it must be written. The solution saves considerable time and maximises the re-use of existing software.

5. OPTIMISATION

5.1. Optimisation of system functionalities

System performance was one of the priorities in the development of the ZSMU FCS software. Performance, responsiveness, and maximum reduction of the main loop processing time became the primary criteria for the applied optimisation. The optimisation largely concerned the building of the application binary version and the applied libraries. In terms of the libraries, the FCS application was originally integrated with the operating system as shown in Fig. 7.

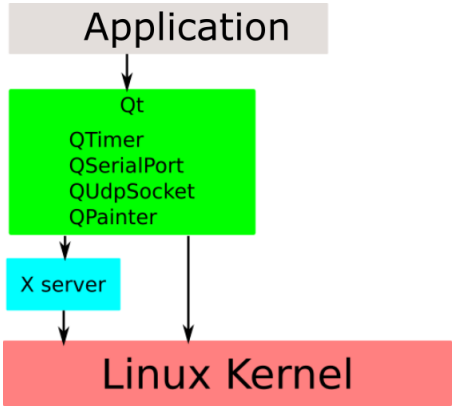

Fig. 7. FCS application and libraries

The first completely proprietary ZSMU design of ZMT S.A. was the ZSMU A4. Its FCS application ran on a Qt library which facilitated the programming of graphical interface features, serial ports, time dependencies, network support, and other features.

However, this solution failed the test of time. The Qt library is cross-platform. To work in different operating systems and on different hardware architectures, the Qt library must form a certain layer of software abstraction. The processing overheads imposed by cross-platform support, double buffering, and the graphical performance in the QPainter class caused many problems. Another issue was the lack of any control over the main application loop: it was embedded in the library core. All the above caused significant variations of the calculation performance time and an unsatisfactory CPU usage level.

The problem is pronounced in applications which require real-time control. The solution was a gradual phasing out of the Qt library in favour of Linux kernel system calls [3]. Currently, the Qt library is only applied for graphics rendering. A dedicated library which handles Linux kernel system calls was developed to handle time, serial ports, network, and frame grabber image capturing. The complexity of the kernel, the small number of examples and inferior or missing documentation posed a challenge for library developers. However, the development outcomes were very good. The direct application of system calls, while omitting all libraries, provided very good time relationships and a markedly reduced CPU usage. Currently, the application communicates with the operating system according to the flow chart shown in Fig. 8.

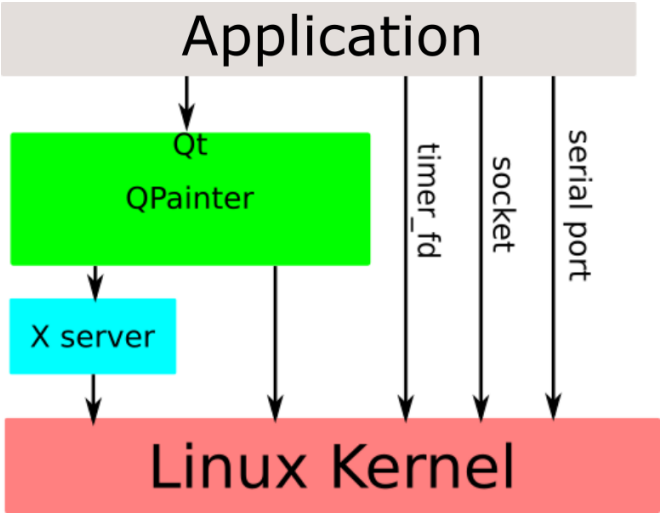

Fig. 8. FCS application and libraries

Figures 9 and 10 show the results of stability testing of data transmission speed over the serial ports and the time of the application response with the output frame to the input frame.

Figure 9 shows the performance result of the test application which transmitted data to a serial port in steady 10 ms intervals and used Linux kernel system calls only to handle the time and the serial port. The run did not suffer from any noticeable time fluctuations. Figure 10 shows the result of the test of the application response to the transmitted data.

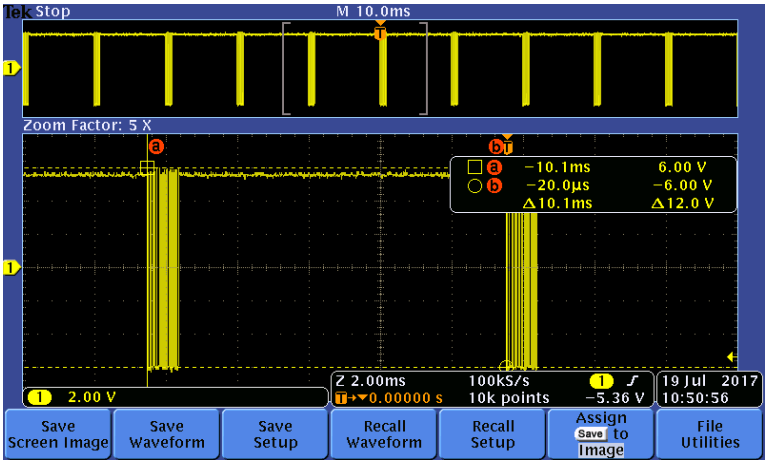

Fig. 9. Clock speed test

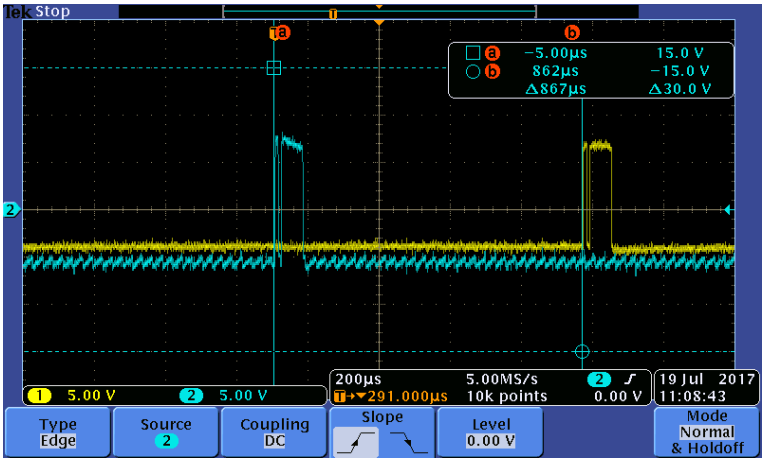

Fig. 10. Response time

The results were highly repeatable and stable, oscillating near 860 µs. Such good relationships were largely due to the Linux system kernel and the solutions it was provided with. While the Linux system kernel is not a real-time kernel by design, many of its aspects meet the requirements defined for a soft real-time system [4].

There are plans to completely eliminate the Qt library. Only the QPainter module is still used to handle the GUI rendering; however, it will ultimately be replaced by an OpenGL or Vulkan – with EGL to generate the rendering context (a software interface to link the application and the graphics card). The planned solutions are superior in that the planned libraries support hardware acceleration of graphics cards and will reduce the CPU load. The X server and the window manager will also be eliminated to reduce the FCS boot time. Currently, suitable libraries and technology demos are in development. The first tests are completed with promising results. Figure 11 shows the final layout of the system application.

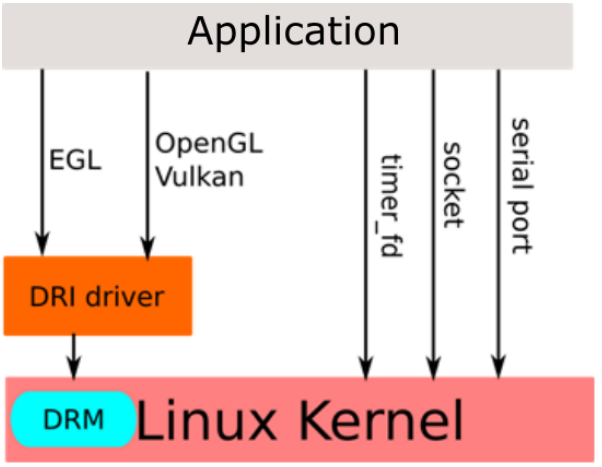

Fig. 11. Final layout of the application in the operating system

5.2. Optimisation of the compilation process

The optimisation included the process of building the final application and operating system. For example, the execution time of the drive control process main loop (ballistics and stabilisation calculations, controllers, etc.) with the default IDE settings was approximately 260 μs. Following the input of compiler settings -O3 and -march=core2, the main loop execution time was reduced by approximately 40 μs, which meant that the data processing speed was 25 kHz. The parameter -O3 enabled the highest degree of source code optimisation. The parameter -march=core2 made the generated binary version of the application fit closely to the CPU it is run on (a Core2Duo chip) and use special instructions of the CPU.

A compilation of the system kernel, system drivers and the standard C and C++ library is planned with the parameters -O3 and -march=core2.

6. TEAMWORK

The development team included 3 to 7 programmers, depending on the product project. The main system was written by two programmers, with one of them tasked with the GUI process and the other one tasked with the drive control process. The stabilisation and ballistics algorithms were also developed by a team of two programmers, one tasked with the algorithms and the other one tasked with the implementation. Two programmers developed the electronic system's software. The whole development process was managed with Git version control.

7. CONCLUSIONS

The ZSMU-1276 A4 remote controlled weapon stations developed in 2014 took ZMT S.A. to a new area of competence. In a period of four years, new software was developed and became a complex management system of all ZSMUs manufactured by ZMT S.A. since 2014. The system is a high-performance, flexible, and stable software solution. It uses the best design practices currently applied in software engineering and fully utilizes the capabilities of the latest Linux operating system. Every component of the system uses hardware acceleration where possible, which optimises hardware resources. The correct performance of the system proves that ZMT S.A. followed an optimum direction to create its proprietary software for the ZSMUs.

FUNDING

The paper contains the results of the research work financed solely by Zakłady Mechaniczne "Tarnów" S.A.

REFERENCES

[1] https://pl.m.wikipedia.org/wiki/C_(język_programowania) (2018)
[2] Prata Stephen. 2012. *Język C++. Szkoła programowania.* Gliwice: Wydawnictwo Helion.
[3] Love Robert. 2014. *Linux. Programowanie systemowe.* Gliwice: Wydawnictwo Helion.
[4] Love Robert. 2014. *Linuksa. Przewodnik programisty.* Gliwice: Wydawnictwo Helion.

Oprogramowanie systemu kierowania ogniem zdalnie sterowanych modułów uzbrojenia – historia, teraźniejszość oraz perspektywy rozwoju

Rafał KRUSZYNA

Zakłady Mechaniczne Tarnów S.A.
ul. Kochanowskiego 30, 33-100 Tarnów

Streszczenie. W publikacji przedstawiono ewolucję oprogramowania systemu kierowania ogniem zdalnie sterowanych modułów uzbrojenia ZSMU, opracowanych i produkowanych w Zakładach Mechanicznych „Tarnów". Opisano architekturę, zakres i cel przeprowadzanych prac badawczo-rozwojowych oraz dalsze potencjalne kierunki rozwoju. Jako uzupełnienie, podano metody optymalizacji zastosowane przy rozwoju oprogramowania oraz organizację pracy dla wieloosobowego zespołu programistów, pracującego nad kodem źródłowym. Publikacja obrazuje, w jaki sposób konsekwentne prowadzenie prac badawczo-rozwojowych prowadzi do zwiększenia efektywności, skalowalności i optymalizacji oprogramowania systemu kierowania ogniem dla zdalnie sterowanych modułów uzbrojenia.

Słowa kluczowe: inżynieria oprogramowania, system kierowania ogniem, oprogramowanie