Cross-Platform Programs Implementation for Specialized Distributed Embedded Systems

Vladimir Sharov, Oleg Bolshakov
Soloviev Rybinsk State Aviation Technical University
Rybinsk, Russia
sharov@rsatu.ru, bolshakov@nppsatek.ru

Alexander Petrov
NPP SATEK plus LTD
Rybinsk, Russia
petrov@nppsatek.ru

***Abstract*—Disadvantages of existing approaches of cross-platform software development for specialized distributed embedded systems are revealed, a new approach to cross-platform programs development in Embeddecy language is disclosed, the approach is based on the generalized model of a microcontroller, an example of the implementation of the analog-to-digital converter control module of the microcontroller with division of hardware-dependent and hardware-independent functions is provided.**

I. INTRODUCTION AND RELATED WORKS

Diversity of types of electronic components is one of the modern trends in embedded systems, meaning both diversity of types of these devices as well as diversity of models of same microcontroller type. The high degree of heterogeneity is one of the key features of specialized microcontroller based embedded systems, which significantly affects the technology of software development, making it more complicated, increasing implementation time and negatively affecting the software mobility. The using of low-level tools does also compound the software mobility problem and has ultimately led to the market state, where embedded systems manufacturers to a large extent depend on the availability and skills of software developers. When a move to a new range of microcontrollers is needed a specialists retraining is required and that leads to an increase in the appearance of additional terms and costs of production. In this regard, the actual problem is the development of cross-platform solutions that would enable effective programming in a heterogeneous computing environment, smoothing the transition from one microcontroller family to another and reducing embedded systems manufacturer's dependence on a specific equipment manufacturer.

The dependence of the code from the computation model is not only connected with the instruction set properties of microcontroller core, but also with its architecture - the presence or absence of certain peripheral modules, as well as their features within the same family of microcontrollers. In practice, the development of specialized embedded systems usually do not require full automatic code portability, which in the "big computers" world is achieved, for example, by the presence of additional runtime or JIT-compilation. This is because of the specialization of considered embedded systems.

It should be noted that the cornerstone of cross-platform software solutions are cross-platform library modules and instrument's architecture that contributes to the degree of connection between the device-dependent and device-independent modules.

Currently, the most common approach that provides some degree of portability for embedded systems, often only within a single family of MCUs, is using preprocessing tools: macros and conditional compilation directives [1]. This enables efficient code, but turns the program into a "patchwork quilt": program logic is broken into multiple fragments, which reduces the transparency of the program code and increases development complexity. By itself, preprocessing is not an extension of the language; it only allows you to represent the code only as a set of character strings and to make replacements in codes as a text without taking into account the code structure.

Another approach is the use of virtual machines to implement execution of a unified code [2]. This approach provides the best solution in portability but involves the use of a virtual machine with heavy resource consumption, which makes it inefficient for specialized embedded systems based on microcontrollers. For example, the implementation of a virtual machine java-NanoVM for Atmel microcontrollers allocate 8 Kbytes of program memory [3] which is a full amount of onboard memory most microcontrollers of this family have.

The study [4] proposes to use a unified assembly language and cross-compiler for different types of systems. In this case, the programmer is offered a low-level tool, which also negatively affects the efficiency of software development for target microcontroller devices.

The most effective of the known approaches to cross-platform software libraries development is based on generic programming paradigm. Under this paradigm developer creates program modules and functions which are parameterized (parametric polymorphism), wherein the parameterization is static, i.e. is implemented at compile time.

This concept is present in some object-oriented programming languages such as C ++ and is implemented as the template mechanism (template classes and template functions). However, for example, when declaring on C++ language a type parameter of some template it is not possible to set any limit on the list of possible types which may be specified. In addition, problems arise in the implementation of flexible control module, capable of working with some configurable output of the microcontroller (pins): the developer either have to use additional resources for storing register numbers matching port addresses, or use preprocessing.

In [5], the class of specialized embedded systems based on microcontrollers is considered, a three-level model of program representation is described, allowing to increase development efficiency through low-level synthesis of elements based on high-level description. Embeddecy language is developed, which is based on the model and can be considered as an extension of C language.

This article considers the method of increasing cross-platform properties of software for heterogeneous embedded microcontroller systems by means of separating the platform-specific software features from platform-independent ones, the method is implemented in Embeddecy language.

II. Generic MCU model

Code portability between different MCUs is implemented in Embeddecy by abstracting from the details of MCUs functioning and by defining the common features. Therefore, one of the priorities in solving the problem of cross-platform is to create a generalized model of a microcontroller.

Fig. 1 shows a generalized model of specialized microcontrollers of most popular families: AVR, PIC18 and STM8. The model describes a generalized architecture of these families microcontrollers, which includes core modules, that are present in some models of MCUs. Interface control modules include UART, SPI, I2C, 1-Wire, USB, Bluetooth, CAN, Ethernet; general settings module include, for example, a watchdog timer, frequency control, microcontroller event system and other functions, affecting the work of the whole chip.

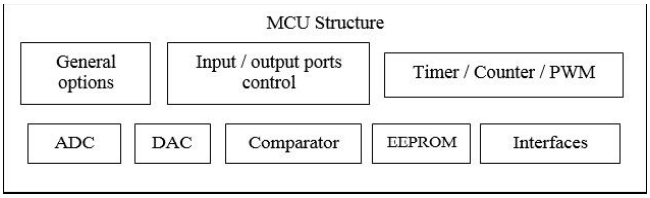

Fig. 1. Generalized model of a microcontroller

Presented in Fig. 1 unified control system modules can be divided into hardware-dependent and hardware-independent, and it should be pointed out that the hardware-independent ones are located at higher levels of abstraction. Fig. 2 shows the diagram of distribution of software modules for the control system levels.

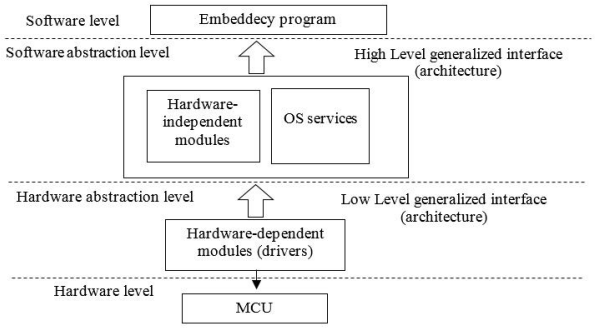

Fig. 2. Levels of control system, with hardware-dependent and hardware-independent modules

Software modules in accordance with their purpose are located on two levels - level software abstractions (SAL) and the hardware abstraction layer (HAL). HAL-level modules are implemented in hardware-specific code that interacts directly with the hardware modules of the microcontroller. Working with the modules at this level is done either through the generic interface (and such instructions are portable), or specific features of a particular module are used (such instructions are unportable). SAL level includes a hardware-independent modules: algorithmic modules or modules emulates the missing hardware modules, such as the USB or I2C virtual control module, and operating system services. Hardware-independent modules can interact with each other or with the device-dependent modules by the generalized interface. In terms of cross-platform, the greatest interest is the interaction of HAL and SAL modules that is specified using the generalized low-level interface. The mechanism of creation of low-level generic interface implemented in a language Embeddecy.

III. Multiple level model of distributed program for embedded systems

To solve the discussed problems arising in the development of distributed software for embedded systems a new multi-level model is offered (Fig. 3).

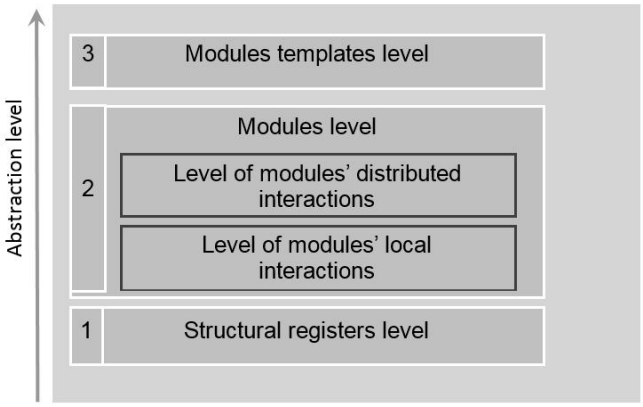

Fig. 3. Level of embedded systems distributed program model

The lower level corresponds to the model of structured programming and allows for compatibility with the above model, the other levels are the superstructure, providing new levels of abstraction.

On the lower level device firmware is described as instructions for configuring peripheral registers' bits - specific memory of the microcontroller. Bit assignment of the peripheral registers are interpreted by the electronic circuit devices, and thus the program is implemented by the microcontroller. According to the model of structured programming, instructions for the registers described in series and grouped into subprograms with parameters. In addition to the instruction register can also read and write the values of the variables.

The second level (the level of the modules on Fig. 3) contains a new kind of abstractions - modules. The modules are units of high-level program in the framework of the proposed model. Developer can use two kinds of modules: tasks and packages that combine all the basic elements of the model. The packages in the model are identical to the concept of packages in Ada or modules in the structural programming languages, so the most interesting are the problem, which will be considered further. Thus, the second level of model is given us a set of tasks.

$$Tasks = \{task\},$$

where the element of the set is an ordered tuple, consisting of the names of the task, a set of functions that can perform the task, signature set of messages that a task can take, and a set of events that can occur in the problem:

$task = <name_{task}, Vars_{task}, Funcs_{task}, MesSigs_{task}, Events_{task}>$,

where $name_{task} \in ID$ – name of *task* (identifier),

$Vars_{task}$ – a set of internal variables of *task*
$Funcs_{task}$ – a set of *task's* functions
$MesSigs_{task}$ – a set of events' signatures of *task*
$Events_{task}$ – a set of events of *task*

The program in the proposed model has a static structure, ie, the user in the design phase determines the number and composition of the tasks to be implemented in each device of a distributed system. This limitation is due to the peculiarities of microcontrollers as the computation units and the specifics of their tasks (task management systems from a static structure of nodes). In addition, it avoids the costs associated with the implementation of intensive algorithms required to support dynamic structure of control systems.

If the package is a simple union of variables, functions, and event structures, the task also includes the control flow is executed in parallel with other tasks and contains a list of signatures received parameterized messages and instructions on how to adopt and send messages. Each task can receive messages, send a message to another task and initiate the event with parameters.

Interaction with the task is carried out only by messaging her. Thus, the task is not only a module in terms of structure, but also parallelism - all task's functions can only be called from task's control flow. Importantly, the modules in the proposed model are static because at the program start moment it's already known the number and composition of modules.

Another important difference between the tasks and the known models is that the interaction between tasks can be carried out not only in sync with the use of "rendezvous", as is done in the models considered, but also asynchronously. In this kind of interaction module does not wait for confirmation of acceptance of its message to more effectively organize the interaction tasks.

The second level of the model consists of two sub-levels: the level of local modules and the level of distributed modules. The level of local modules allows us to describe the program in the form of interacting modules within a single device. The level of distributed module allows to describe the program in the form of interacting modules located on different devices. In addition, at the distribution level it comes abstraction from the properties of the passive / active equipment and communication protocols: it can be assumed that each device can be active, and any of its modules can initiate sending messages to the module of other device and at the same time for the implementation of communication protocols meet the specific interface modules with a known set of signature features that they can use.

At the third level of the model is a set of templates

$$Templs = \{templ^m\},\ m \in \{1..k\},\ k > 0,$$

where m – the dimension of the template defined by a limited number of its parameters, and the template is a tuple consisting of the name of the template and the mapping function of a fragment of the instructions and template settings on multiple tasks.

$$templ^m = <name_{templ}{}^m, Finst^m>$$

$$Finst^m : C \times Params_{templ}{}^m \rightarrow Tasks_{inst}$$

$$Tasks_{inst} \in Tasks$$

$$Params^m = Param_1 \times \ldots \times Param_m$$

Templates are static and have a set of parameters, at the moment of setting specific values of which (a template instantiation moment) template becomes a module. There are three kinds of template parameters: text-substitution parameter, device pin parameter and module parameter. Thus, the templates allow you to create universal modules that can work with different pin devices and access a number of third-party modules. This flexibility built into the templates provided by the static specifying parameters. Support systems static structure allows you to specify a priori information about the program, thereby reducing the amount of memory used by the calculator and the number of actions required to maintain the integrity of the dynamic structure and increase the level of

abstraction in the development of distributed programs.

IV. Templates in Embeddecy programming language

Embeddecy programming language can be considered as an extension of the C language specialized for the development of distributed embedded systems software. The main motive of creating a new language was unsatisfactory balance between efficiency and abstraction of existing programming languages used in the development of embedded systems software. Analysis of typical problems in this area, as well as the emergence of alternative instruments, confirmed the relevance of the development of the programming language that combines the effectiveness of the code in C language and high-level concepts of C# and Java languages. One successful example of combining high level abstract constructs and efficiency can be regarded as the language Ada, which, however, in practice of the development of microcontroller systems is rarely used, in particular, due to limitations in the synchronization software processes means and overly strong typing.

During requirements analysis model was developed by a multi-level representation of the software for distributed embedded systems ([6] and hereinabove). Language Embeddecy includes support for all elements of the model (task packages, templates), and the following language constructs that complement the C language: constructs for parallel programming (sending synchronous / asynchronous, accept the message), namespace, anonymous functions, types, delegates and variables delegates, events, developed tools for describing static templates, keyword "via" to transmit messages between distributed modules, constructs, defining the visibility scopes. The essential feature is the concept of language patterns modules allow you to develop cross-platform language Embeddecy library modules. Template module - it is parameterized module, the parameters that are set at the point in time of compilation. There are three kinds of template parameters: text-substitution parameter, device pin parameter and module parameter.

At the template module instantiation template parameters are set and the template code is converted into a code module by substituting actual values instead of the names of the formal parameters. For example, if developer is using the pin-parameter in the description of module template, then at the moment of template instantiation the generation of code for writing and reading data from mapped output device memory bit occurs. The language also allowed partial template specialization of such a mechanism similar to language C++. In this case, based on the template with multiple parameters create another pattern in which a part of the formal parameters specialized actual values and part of parameters (at least one) remains formal.

Format description of the template and its parameters is shown below:

template <module_type> <*name*>
<
macro <*param_macro_name*>,
type <*param_type_name*> **is** <type1 > | <type2 > | … | <typen >,

pin <*param_pin_name*>,
value <*expr_type_name*> <*param_expr_name*>,
module <*interface_name*> <*param_module_name*>
>

{
<...> // module template's body
},

where *module_type* – type of module: package or task, *param_macro_name* – macro parameter name, *param_pin_name* – pin parameter name, *param_expr_name* – expression parameter name, *expr_type_name* – type of expression parameter name, *param_type_name* – type parameter name, *param_module_name* – module parameter name, *interface_name* – name of interface, that template should implement.

V. Using templates of modules' interfaces for the explicit separation of portable code

One of the goals of the interfaces introduction to the Embeddecy language is the separation of hardware-dependent code and hardware-independent. Using this mechanism allows you to create a system library modules, using which the developer will have a clear idea of which code is portable and which is not portable.

Way of library modules development based on proposed language is described below.

1) For each type of hardware-dependent module in the model of generalized microcontroller it should be developed a standardized template interface with a list of common to all of these modules functions and data types.
2) Hardware-specific modules templates are developed for each model of microcontroller that implements a generic interface. At this moment a partial specification of template generic interface could be useful.
3) Application firmware developer instantiate in a code a hardware-dependent module based on the desired hardware-dependent pattern. To access the module code in cross-platform way, you need to contact him via the generic interface.
4) Then developer can declare a hardware-independent modules and set as parameters value hardware-specific modules, to work with it via a generic interface.

Below a mechanism of separation cross-platform code from non cross-platform code is considered as the example of ADC control module implementation for AVR and STM microcontrollers.

In this example, cross-platform interface template *ADC_GENERAL* and template of hardware-dependent ADC

control module (*ADC_STM*) are described on the Embeddecy programming language. To ensure portability we create an alias (*adc_crossplatform*) for all reference to hardware-dependent modules. All reference to the data structures and functions of the module via an alias *adc_crossplatform* are cross-platform. If you want to use specific functions of the equipment, then refer to these functions must be through hardware-dependent alias (in this example - *adc_hardware*), ie through the name of the module, instantiate directly on hardware-dependent pattern. At the same time to a function and data structures can become importable. Worth mentioning that the introduction of the super types concept to language Embedded, give us a possibility to limit the possible type that can be passed as a parameter template. For example, when describing a generalized interface module for ADC (*ADC_GENERAL*) stated that as the *ValueType* type can be selected either integer (*int*) or *unsigned byte* type. This approach greatly simplifies the use of templates for the development of libraries of software modules.

```
/* generic interface for ADC control module */
template interface ADC_GENERAL

<module MCU_GeneralConfig generalConfig,
type ValueType is int | unsigned byte,
type BitsResolution_enum is
ADC_AVR.BitsResolution_enum |
ADC_STM.BitsResolution_enum |
ADC_PIC.BitsResolution_enum, ...> {
  void Init();

  <...>
  ValueType ConvertGetSample (Channels_enum channel)
  <...>
}

template package ADC_STM

<module MCU_GeneralConfig generalConfig,
type ValueType is int | unsigned char> implements
ADC_GENERAL

<module MCU_GeneralConfig generalConfig, type
ValueType is int | unsigned char, BitsResolution_enum =
ADC_STM.BitsResolution_enum, ...>

{
        public typedef enum BitsResolution {…};
        <...>
// ADC_GENERAL function implementation
        <...>
// functions below are importable
        public bool getOverrunInterruptEnabled();
        <...>
}
<...>
// Now we could instantiate package for work with STM ADC
module
#define_module stm8l152c6cfg =
STM8l152c6_GeneralConfig<...>
#define_module adc_hardware = ADC_STM
<stm8l152c6cfg,unsigned char> ;
#define_module adc_crossplatform = (ADC_GENERAL <
generalConfig, ValueTYpe, BitsResolution_enum =
ADC_STM.BitsResolution_enum>, ...) adc_hardware;
<...>

// cross-platform instraction
adc_crossplatform.Init();

// not cross-platform instraction
adc_hardware.getOverrunInterruptEnabled();
```

VI. Conclusion

The problem of creating software for specialized heterogeneous distributed embedded systems based on microcontrollers is described in the paper. The relevance of cross-platform software development technology is stated. Analyze of existing approaches to the problem of software portability is shown. A generalized model of microcontroller underlying the proposed approach to the development of a unified library of software modules in the Embeddecy language, providing a higher level of code portability is presented. Increasing the level of code portability is reached by means of using Embeddecy language facilities such as ability to describe on high level parallel behavior and synchronization of distributed modules. An example of the implementation of cross-platform template module developed on Embeddecy language is considered.

The next stage of our project will be the implementation of the language translator and low-level code generation to realize parallel modules execution both between modules in one microcontroller and between ones distributed between devices. It's evident that the code generation will lead to some level of redundancy but the main idea that that redundancy will be much less significant than known approaches may suggest, leading to high increasing in development efficiency. The results of the integrated development environment including code generation tool are going to be published on the project website (http://www.mcublocks.com) in near future.

Acknowledgment

This work was supported by the Ministry of Education and Science of the Russian Federation (No 14.607.21.0012 (RFMEFI60714X0012) agreement for a grant on 'Conducting applied research for the development of intelligent technology and software systems, navigation and control of mobile technical equipment using machine vision techniques and high-performance distributed computing')

References

[1] I. Paramonov, A. Vasilev, D. Laure, N. Kozhemyakin, "Preprocessor Based Approach for Cross-Platform Development with Qt Quick Components", *in Proc. of the 11th Conference of Open Innovations Association FRUCT, St-Petersburg,* 2012., Web: https://fruct.org/publications/fruct11/files/Par.pdf

[2] A. Platunov “Theoretical and methodological foundations of high-level design of embedded computing systems”, abstract PhD: 05.13.12., СПб., 2010., 39 pp.

[3] The NanoVM - Java for the AVR, Web: http://www.harbaum.org/till/nanovm/index.shtml

[4] A. Belih, “Unification single chip architecture and its application to the development of embedded software”, abstract PhD: 05.13.15. , M., 2006., 20 pp.

[5] O. Bolshakov, A. Petrov, “A model of distributed microcontroller governmental systems software”, *in Proc. Theory and practice of systems analysis, Rubinsk, RSATU,* 2014, 96 – 107 pp.

[6] O. Bolshakov, V. Sharov, A. Petrov, “Model of software programs for distributed embedded systems”, Herald of Rybinsk State Aviation Technical University named after PA Solovyov, Rybinsk, 2015.

[7] D. Cheremisinov, "Design and analysis of parallelism in processes and programs", Minsk: Belarus, Nauka, 2011. – 300 pp.

[8] E. Dijkstra, “Notes on Structured Programming”. — M.: Mir, 1975. — 7–97 pp.

[9] Can I use C++ on the AVR?, Web: http://www.nongnu.org/avr-libc/user-manual/FAQ.html