Blended Approach in the Embedded Software Design

Ivan Georgiev Buliev, Teodor Borislavov Grigorov and Jordan Nikolov Kolev

Abstract – **The main purpose of the proposed approach is to facilitate and accelerate the embedded software development using a multiplatform software development environment and the development of virtual devices drivers for screen visualization and interactive models for I/O components of an embedded system. The whole development may take place on the PC in a simulation mode. Finally the virtual drivers are replaced by the actual subroutines for the real devices and the software is recompiled, programmed and run in the embedded system itself.**

Keywords – **Embedded software development, virtual drivers, software simulation**

I. Introduction

A. Challenges, faced by the embedded SW design

Embedded systems are everywhere around us – from mass products like smartphones and numerous other gadgets, through commercial, industrial, and military applications, to high-level scientific and research instrumentation. Regardless of the application specifics, the embedded systems have many common features and meet more or less common requirements. Among them are: real time implementations, flexible specifications, versatile platforms, multiple periphery, harsh environment, short time-to-market, cost restrictions, user friendly interface is a must (where applicable), highly competitive market, etc.

Although the traditional design flow and available tools for hardware and software development are continuously refined, the need of further improvement in the field is out of doubt.

B. Traditional design flow and tools

During the years, for various reasons, the C programming language has been widely accepted as the high-level language for programming Embedded systems. The traditional embedded software design usually involves a kind of Integrated Development Environment (IDE). The IDEs usually offer a plain text editor for code writing, extended with language-dependent syntax highlighting, an optimised C/C++ compiler, and a debugger able to connect and "speak" to the target system via different types of programming and debugging interfaces. Commercial products offer fast and convenient software design flows. Open source IDEs and compilation and debugging toolchains are also available. The usual sequence of actions is shown in the diagram in Figure 1.

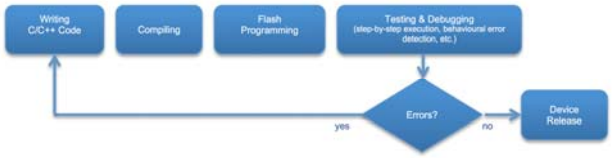


Fig. 1. Traditional design flow of the embedded software development.

Although the process seems straight forward, a number of particularities usually hinder the fast development and require careful and sometimes very detailed testing and step-by-step debugging. The compilation takes time. The programming of the flash memory also takes time and depending on the complexity and the program size, this time may be significant. Organising the debugging, setting up the break points, the code execution and the step-by-step execution through the various debug interfaces (JTAG, SWD, BDM, HID-based, etc.) are in general also time consuming. In some cases, as for example applications with externally imposed real-time functionality, the debugging is not only difficult but also even impossible.

In comparison to the programming for personal computers, perhaps two main differences can be distinguished. The embedded systems monitor and control separate either single-wire or grouped number of digital inputs and outputs but not necessarily multiples of eight, while the smallest container in ANSI C is the character, i.e. the byte. The other particularity is the presence in the personal computers and the absence in the embedded systems of the standard input/output console and the keyboard. Based on the teaching experience of the authors, these two differences are in the same time the main challenges in front of the young embedded software developers, usually having experience in programming personal computers.

In the same time, those who understand and adopt the differences start combining the two types of experience in their favour. Some advanced concepts from the personal computer software domain seem partially applicable in the embedded software design and presented below.

C. Attractive concepts from the personal computer programming

The personal computer programming continuously generates new ideas and concepts. Normally, they concern and are applied for computer systems of higher complexity, running operating systems (OS). Some of them however seem attractive for the embedded software design, as well.

I. Buliev is with the Department of Electronic Engineering and Microelectronics, Faculty of Electronics, Technical University of Varna, 1 Studentska str., 9010 Varna, Bulgaria, e-mail: ivangbuliev@gmail.com

T. Grigorov and J. Kolev are with Micon K Ltd, 42A Papadopulu str., 9010 Varna, e-mail: jkolev@ieee.bg

The application-driver model

Using of drivers is usually related to applications, which are run from a given operating system. This is also valid for the embedded software development for more complicated designs [1]. However, simpler embedded system designs could also benefit. If their firmware were structured appropriately, changing a single system component would not need much effort. For example replacing a monochromatic LCD with a true colour RGB display would be done fast, if the display output functions are based on a limited set of low-level functions, which will only need to be replaced. This is actually the generic idea of using drivers.

Virtualization

Virtual device drivers are a particular type of device drivers [2], [3]. They are used to replace and emulate real hardware device, especially in the quite popular recently virtual environment. In such an environment, a file can be interpreted as a CDROM drive with inserted disk in it for example. One could perhaps not find immediate application of the virtual drivers in an embedded system programming but we actually developed this idea and we present some results in the next sections of this paper.

Multi-platform SW development environments

The times when the only programming language for embedded systems was Assembler are far in the past now. Developers still use it but the high-level programming languages give much greater flexibility today. Among them, the C/C++ language is undoubtedly recognized as the most popular. The ANSI version of the C/C++ language is usually extended by the manufacturers to better meet the microcontroller particularities. Missing data types, as for example – bit, are usually added in a non-standard way. This makes the developed C/C++ application code actually non-portable and moving from one architecture to another is not straightforward.

Recently multi-platform compilers and libraries have been developed and offered as commercial products. Limited but open source versions are also released. One such platform that allows OS-independent application programming is Qt [4]. Again the presence of an OS is a prerequisite, and direct using of Qt for embedded programming is not worth in case of simple applications.

The advantages of the three above-mentioned technologies were exploited. In the present paper we present the developed by us approach for faster embedded software development and verification and we provide some successful examples.

II. Blended Approach for Embedded Software Design

A. Combining advanced practices

The main idea in our approach is to transform to a large extent the embedded software development into a software development for a personal computer.

One of the basic differences between the embedded systems and the universal computers is in the variety and the number of peripheral devices. A common personal computer has a screen as an output device and a keyboard and mouse as input devices. The embedded systems can have as I/O devices buttons, LEDs, LED-based indicators, intelligent LCD displays, touch screen panels, etc. but they may have also a plenty of sensors connected and providing different type of data.

No matter how many and different the peripheral devices are, almost always they can be attractively visualized on the screen of the personal computers. The input devices can be emulated with the help of the keyboard and the mouse. The sensor outputs may be modeled as sliders, the relay or transistor outputs may be manually switched on or off by clicking on small push buttons, etc.

B. Practical virtualization in the embedded SW design

Therefore, we suggest using of a multiplatform software development environment, allowing programing of graphical user interface (GUIs) applications and the development of virtual devices drivers and screen visualization and interaction models for the different I/O components of an embedded system. Then the whole development may take place on the PC in a simulation mode. At the end the virtual drivers are replaced by the actual subroutines for the real system devices and the software is recompiled, programmed and run in the embedded system itself. Using ANSI C/C++ for the common part of the code is a requirement but it is not a significant limitation in comparison with the significant acceleration in the software development process, by avoiding the continuous time-consuming uC reprogramming and stiff debugging. Conditional compilation using different keys for the PC and the embedded system provides the means for creating even a common project. The diagram in Figure 2 illustrates the main concept.

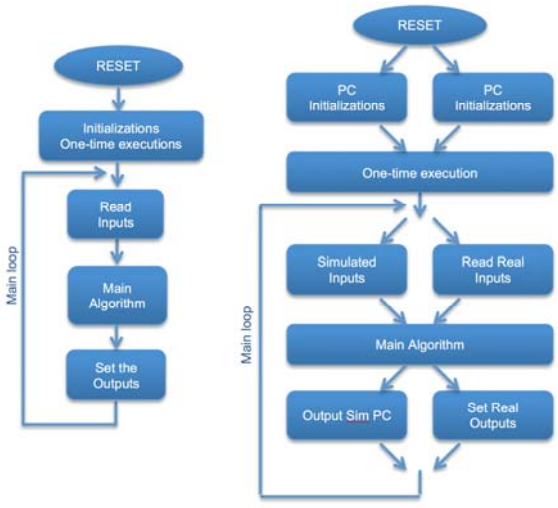


Fig. 2. Diagram, presenting the main concept

One time-consuming task in the software development is the organising and the rendering the various GUI screens. Using the virtualization approach we could cope much faster that initially planned. Figure 3 shows one of the GUI

screens as designed in the PC application. Figure 4 shows the real screen of the manufactured equipment.

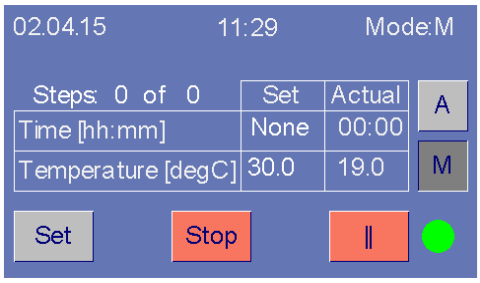


Fig. 3. A screenshot from the PC simulator application

Fig. 4. The same screen from Fig. 3 on the display of the real device.

The main slow-down however in the software development for this task was due to the duration of the real experiments. Significant time was needed for the real cooling or warming of the chamber. What we did was to virtualise the sensors and simulate their output with the help of common graphical controls in Windows – sliders, buttons, etc. (Figure 3).

This allowed to faster reach (although simulated) the conditions, requiring system response and thus provided means for much faster behavioural algorithm testing.

The project trees for the PC simulator application and the embedded application follow the concept from Figure 2 and are shown in Figure 5a.

The projects use the same header files and the same C source files. In Qt the C source files are #included within the C++ source files. The project files for Qt and MPLABX are absolutely independent. Specific and different C compiler keys were defined in each of them. This allowed the inclusion/exclusion of code segments in the common files depending on the platform for which the compilation took place. Figure 5b illustrates the conditional code compilation in case of PIC microcontroller or personal computer.

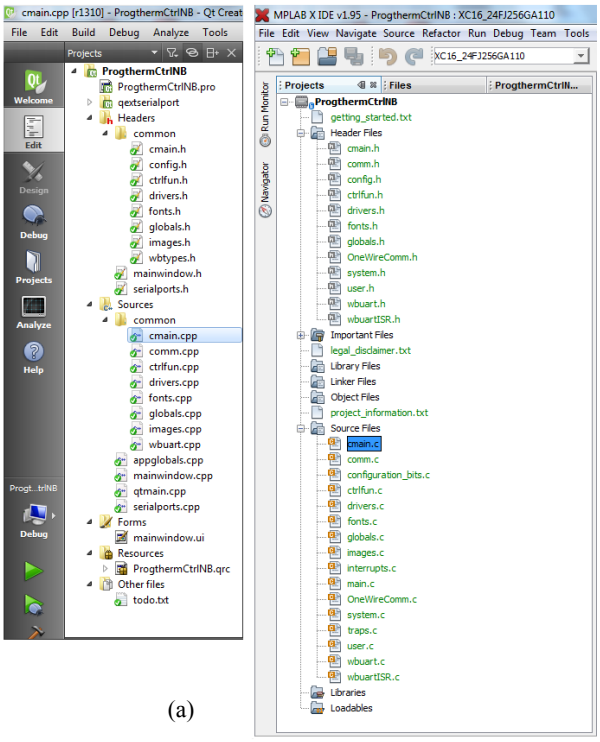
Porting the software from the simulator to the real device took about half a day. Final software refinements took few more days. Thanks to the applied blended approach, a significant reduction in the development time was achieved.

The suggested approach has been successfully applied in three different applications, provided as examples in the next section.

III. Successful Implementation Examples

A. Programmable thermostatic chamber

Physical experiments are performed at a wide range of the ambient temperature. The programmable thermostatic chamber that was developed has a working space of 50 dm^3 where a physical equipment and instrumentation can be placed. The temperature of the working space is controlled versus time in a range from -15^{0}C to +60^{0}C by a program.



(a)

```
// E2P related functions
int WriteE2P(short addr, char n, char *src)
{
   int res = GSUCCESS;

#ifdef WINEBOTSIM
   // QT
   for (int i=0; i<n; i++)
      e2pMemory[addr+i] = src[i];
#else
   // MPLABX
   char   *p, i;
   uint8_t AddrH, AddrL;

   AddrH = addr/256;
   AddrL = addr%256;
   p = src;

   Start_my_I2C3();
   res = Write_address_I2C3(EEPROM|EWRITE);
   res = Send_data_I2C3(AddrH);
   res = Send_data_I2C3(AddrL);
   for(i=0; i<n; i++)
      res = Send_data_I2C3(*p++);
   Stop_my_I2C3();
#endif
   return res;
}
```

(b)

Fig. 5. Project trees for the PC simulator application and the embedded application (a), and part of the code illustrating the different low-level implementation of writing to an I^{2}C EEPROM (b).

The program can be either entered manually at the device LCD touch panel or loaded from external computer as a preliminary prepared file. The number of time-temperature steps can be up to 255 with a total duration up to one month. A log file with temperature and time data can be recorded on an external computer.

B. ECETD

ECETD (Electro-Chemical Etching of Track Detectors) is intended for processing (etching) of general-purpose polycarbon compact discs (CD). The material of the CDs, i.e. polycarbon, being exposed to ionization radiation, particularly caused by Radon decay, keeps the tracks of the resulting alpha particles. The intensity of tracks is an integral estimate of the intensity of the ionization radiation on the place where the CDs are stored. This can be used for monitoring of ambient radiation in living areas. The tracks are very small and can be visualized by using of electron beam microscopy only. The etching process realized in ECETD makes the tracks visible by optical microscopes or even with unaided eyes. ECETD consists of a HV digital synthesizer (up to 4 kV, 6 kHz), electronic control, power supply modules and thermostabilized by Peltier coolers/heaters processing platform. Controlled parameters are: the processing high voltage, etching current, processing platform temperature, and duration of etching. The parameters are using the combination of a coloured LCD and a touch panel on top of it. An activity log can be created on an external computer through a USB connection. The manufactured device is shown in Figure 6.

Fig. 6. The ECETD device.

The software development also benefited significantly from the application of the above-described approach. GUI screens composition, touch screen input and device reactions were preliminary programmed and tested in a simulation mode and later ported to the embedded device.

C. Wine dispensers

Much more complex network system of wine dispensers was developed also with the partial support of the presented approach. Such automated wine dispensers are used in restaurants, wine shops, bars, wine tasting rooms, etc. They offer to the customers the opportunity to purchase small doses wine either for degustation (e.g. 20-30ml) or a full glass for pleasant consuming. The wine is dispensed from original bottles, which are being kept at the most suitable temperature for the respective wine sort. To preserve the wine in the bottles, to avoid oxygenation and prevent it from losing its genuine flavour, taste and colour, the wine is dispensed out from the bottle with the help of low-pressure nitrogen or argon gases. The tasks performed by the embedded controller include: temperature control of bottle section, gas pressure control, volume control of dispensed doses, RFID access control, etc. Main computer keeps a database for the wine bottles installed in the dispensers as well as for those stored in the shop.

Fig. 7. The wine dispenser

The challenge in the software development was not only the design of the GUI screens and the main algorithm programming but also the implementation of the network communication, allowing the simultaneous work of a number of wine dispensers connected to a main computer. The communication protocol was tested and tuned in a simulation environment, using two personal computers, one of them playing the role of the dispenser.

IV. CONCLUSIONS

The successful implementation of the three different products undoubtedly confirms not only the feasibility but also the benefits from the approaching the problems using a combination of simulations and real implementations and testing.

Qt and MPLAB proved to be able to be used in common projects. It would be also of interest to investigate and confirm the possibility to use together for example Qt and Eclipse and GCC for ARM development. Commercial IDEs (e.g. Freescale's CodeWarrior, TI's Code Composer) are based on Eclipse could eventually bring benefits as well.

REFERENCES

[1] *Embedded Systems Architecture, Device Drivers - Part 1: Interrupt Handling*, EDN Network, March 05, 2013, http://www.edn.com/design/systems-design/4408329/ Embedded-Systems-Architecture--Device-Drivers---Part-1--Interrupt-Handling

[2] G. Heiser, *The Role of Virtualization in Embedded Systems*, First Workshop on Isolation and Integration in Embedded Systems (IIES'08) April 1, 2008, Glasgow, UK

[3] A. Kadav, M. M. Swift, *Understanding Modern Device Drivers*, ASPLOS'12, March 3−7, 2012, London, England, UK

[4] *Qt Framework* (http://www.qt.io)