ISSN : 0974 - 7435 Volume 10 Issue 14

2014

BioTechnology

An Indian Journal

FULL PAPER

BTAIJ, 10(14), 2014 [8055-8062]

Research on component-based embedded development engineering framework

Zengwang Yang*, Yihuai Wang, Xinyu Dai1, Si Chen
School of Physics & Electronic Engineering, Jiangsu Normal University, Xuzhou 221116, (CHINA)
School of Computer Science & Technology, Soochow University, Suzhou, 215006, (CHINA)
Email: yangzw72@126.com; yihuaiw@suda.edu.cn; daixy@jsnu.edu.cn, chensism@126.com

ABSTRACT

The framework is an important way to obtain the reusability of software in one field, and the component technology is a proven technical means which can improve the overall benefit in the software life cycle. For the two types of problems existing in the MCU-based development of embedded system, the paper proposes a component-based engineering framework of MCU-based embedded development, details the definition and classification of embedded components and principles and methods of component design, and analyses the engineering framework and its transplantation methods. By means of the tests and applications of different development system, the results indicate that the engineering framework improve software development efficiency and maintainability and reusability, and effectively reduce the technical difficulty and complexity of embedded development.

KEYWORDS

Component; Engineering framework; Embedded development; Reusability; Portability.

INTRODUCTION

From the 1970s the emergence of the single chip microcomputer to now a variety of embedded microprocessor large-scale applications, embedded systems have a history of nearly 40 years of development. Currently, it is widely used in various fields of office automation, consumption, communication, automobile, industry and military etc[1]. In recent years, with the reliability of microcontroller (MCU) internal flash memory improving and its erase ways changing, the capacity of the internal RAM and Flash memory increasing, as well as the built-in degree of external modules improving, the scale and complexity of embedded design have undergone fundamental changes; however, the method of embedded development has not been very good development. At present there are some problems which restrict the further development of the embedded system development based on MCU[2]. first, there are little hardware design specifications of embedded systems. Usually developers design system hardware circuit with personal experience. This leads to poor hardware portability and reusability due to the lack of design specifications. Second, for a long time, because of the support of software engineering idea in the process of programming drivers, software and hardware have been designed separately, leading to the poor universality, portability and reusability of the underlying software, which is closely related to hardware. The lack of a standard, document-oriented management has also imposed much difficulty on the communication among developers and future system maintenance.

Although it is far behind other areas for the software engineering development in the field of embedded system, some scholars have tried to introduce the idea of component-based software engineering into the embedded field in recent years, providing theoretical support for embedded software development[3]. Component technology is a proven technical means which can fully improve the overall benefit of each phase of the software life cycle[4]; Component-based engineering framework is an important and effective way which can improve the embedded software reusability, portability and maintainability[5]. The component models currently used are only applicable to existing operating system software development field, while the component models are researched rarely in the wide fields of MCU-based embedded development without operating system both at home and abroad. To resolve the problem, the paper takes advantage of the collaborative design thought of embedded hardware and software, and puts forward a definition of the low-level component, which abstracts and packages the low-level drivers Closely related with hardware, and further introduces component-based software development technology into the process of MCU-based embedded software development. This paper proposes an engineering framework of MCU-based embedded development, which can realize the application and promotion of component-based technology in the embedded development field without operating system, and can reflect the high efficiency, high reliability characteristics of component-based software development model and enhance the product flexibility, scalability and maintainability[6].

EMBEDDED COMPONENT CONCEPT AND HIERARCHICAL MODEL

Hardware component

Embedded hardware is an integral and important part in any embedded products, and is the building base of the entire embedded system. Embedded application program and operating system run in a particular hardware system. An embedded system that MCU is the core typically includes the following hardware modules: power supply, writer interface circuits, hardware support circuits, UART, USB, Flash, A/D, D/A, LCD, keyboard, sensor input circuits, communication circuits, signal amplification circuits, driver circuits. Some of the modules are integrated inside the MCU, some are outside the MCU.

The module division is based on its function. As is different from hardware modules, the embedded hardware component refers to a reusable hardware entity that is formed by packing one or more hardware functional modules, support circuits and its function description and which provides a series of standard input/output interfaces. According to the definition, the traditional hardware module is an integral part of hardware component and a hardware component may contain one or many hardware functional modules.

According to the production and consumption relationship between interfaces, the interfaces can be divided into two types, providing interfaces and demanding interfaces. Hardware components can be divided into the core components, the intermediate components and the terminal components on the basis of their different interface types. The core components only have providing interfaces but no demanding interfaces. That is to say, they only provide services for other hardware components, rather than receiving services. In embedded systems that a single MCU is the core, the minimum system of MCU is a typical core component. Intermediate components have not only the demanding interfaces but also the providing interfaces, they are not only able to accept the services provided by other components, but also able to provide services for other components. While the terminal components only have demanding interfaces, they only accept the services provided by other components. The differences among the three types of components are shown in TABLE 1[8].

Software component

Embedded software component (ESC) is a group of packed, standardized, reusable software units with embedded character to achieve a certain embedded function, and the function unit to assemble embedded system. Embedded software components are divided into high-level software components and low-level software components (hereinafter referred to as high-level components and low-level components). The high-level component is hardware-independent, while the low-level

component, the driver of the hardware, is inseparable from the hardware. The low-level component contains low-level internal component, low-level peripheral component, and MCU header file. The low-level internal component also contains GPIO component and functional component[7].

TABLE 1:The difference among core components, intermediate components and terminal components

Type	Demanding Interface	Providing Interface	For Instance
core components	no	yes	the minimum system of MC9S08AW60、the minimum system of MCF52233
intermediate components	yes	yes	variable frequency generating components、voltage conversion components
terminal components	yes	no	LCD components、LED components、keyboard components

Hierarchical model of embedded components[9]

The hierarchical model of the embedded hardware components and software components is as shown in Figure 1, it is helpful for us to further understand the relationship between various components. As mentioned earlier in the hardware components, the core component is the minimum system of MCU. Usually, MCU contains GPIO (General Purpose IO) ports and a number of built-in function modules. The driver of the GPIO is packed as the GPIO component, while other built-in function modules' drivers are packed into functional components, such as UART component, Flash component, USB component, IIC component and so on[10].

On the hardware component level, compared with the core component, the intermediate component and terminal component are the "peripheral" of the core component. The software components formed by packing drivers of these "peripheral" is called the low-level peripheral component. Note that not all the intermediate components and the terminal components can be used as programming objects. For example: keyboard, LED, LCD and other hardware components can be used as programming objects, but the electric conversion hardware component has nothing to do with programming, so there is no corresponding low-level driver, and of course, no corresponding software component.

As can be seen from Figure 1, Low-level peripheral components can call the low-level internal component, such as the LCD component can call GPIO component, PCF8563 component (clock component) can call IIC component and so on. High-level components can call low-level peripheral components and functional low-level internal components, but can not directly call GPIO component. In addition, taking into account that almost all of low-level internal components are related to MCU registers, all registers are arranged together to form a MCU header file, so that other components can include the header file[11].

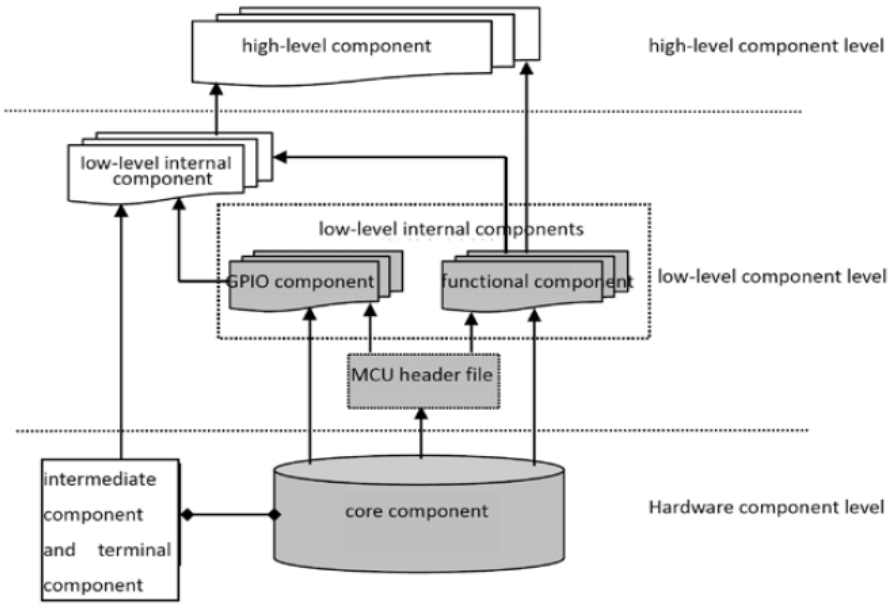

Figure1: The hierarchical model of hardware and software components of embedded system

Low-level component implement method

The low-level components directly communicate with the hardware, which are the core content of embedded development based on components, and are the bridge and the hub connection between components. Usually the low-level components are composed of head file and source file except the MCU header file[12].

Header file contents are: #include statement of low-level component, Macro statement for component property, and interface function prototype statement. Using function prototypes in the header file is helpful for the establishment of the code modules and external interface specifications. When using these functions, users can directly read the prototype of these functions, no need to see the source. The source file contains the definition of the internal and the external function of the component, that is, the implementation code which helps the function to perform[13].

COMPONENT-BASED ENGINEERING FRAMEWORK DESIGN

Design thought and component-based engineering framework organization

According to the thought of software engineering, it must be reusable, portable, and understandable for design and organization of engineering framework, and help to improve the efficiency of embedded software develop-ment and shorten the development cycle[14]. Component-based engineering framework is not only an important way to achieve software reuse but also the software reuse research focus, which inherits the software component technology. When designing the component, it is the most critical task to find commonality and individuality of the component, and extract the properties and external interface functions[15]. When a component applied to different systems, header file of the component is the only to be changed, while source files of the component do not have to be modified, or alter little.

Following these thoughts and basing on the project templates provided by CodeWarrior integrated develop-ment environment, we summarize the classification of common files and directory names of the engineering framework, and improve the program structure by applying the concept of the low-level software compo-nent and then design a new component-based engineering framework SD-NOS. Framework structure organization is shown in Figure 2[16].

SD-NOS engineering framework analysis

Taking the LED flashes (as lights) of simple engineer-ing application as an example, we will introduce the embedded project file organization method based on the CW environment. Figure 2 shows the relevant source files' tree structure, which can be classified into five parts: "Assembler Program"(S), "Header File"(H), "C Pro-gram"(C), "Link File"(link) and project documenta-tion (Document.txt). The "header file" can be divided into general framework header file (Frame_H) and software component header file (Component_H); and "C Program" into general framework of the procedure (Frame_C) and software component (Component_C) accordingly.

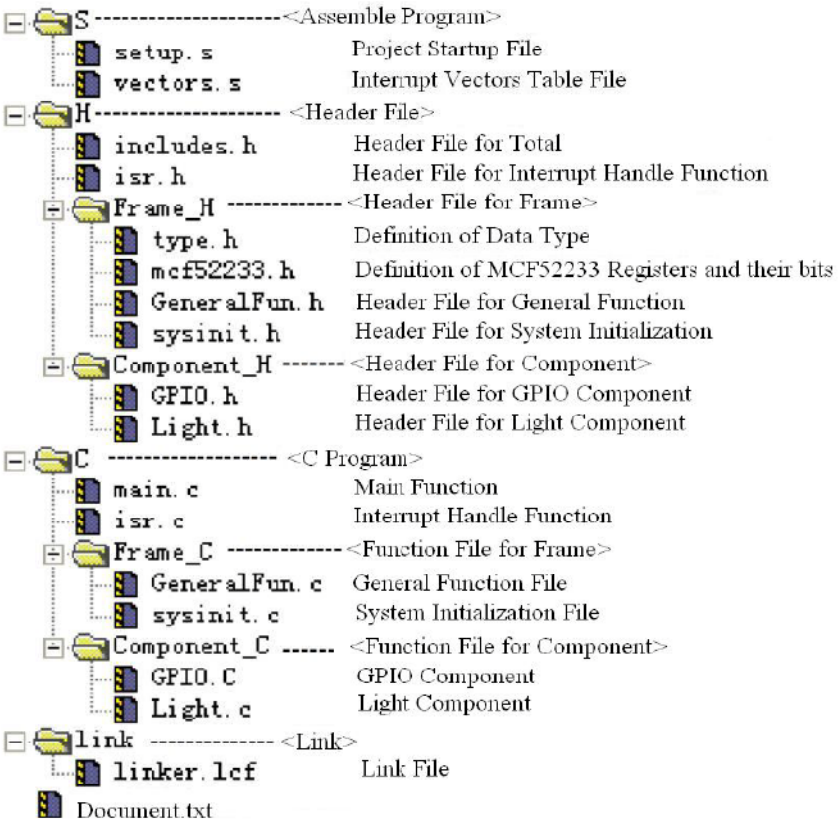

Figure 2: SD-NOS Engineering framework for light project

The file that included in the "Assembler Program " is related to the initialization of the project, including project startup file setup.s and interrupt vector file vectors.s. Because the source code is completely written in assembly language, so

we put it as a class separately. For other documents, it is very easy to find the corresponding relationship between a header file and a source file. For example, includes.h corresponds to main.c, isr.h corresponds to isr.c, and GPIO.h corresponds to GPIO.c. They are cut and organized in accordance with the component-based principle.

Please pay particular attention to two documents: main.c and isr.c. From the point of source file, the general implementation flow of embedded software is as follows: system starts-up and initializes, the program sequentially implements in accordance with the main loop that defined in main.c; when confronted with interrupt request, it will implement the interrupt handling procedures that defined in the implementation of isr.c after completing the interrupt handling, it will return to the interrupt place to implement the procedure in order. Because main.c and isr.c reflect the overall implementation process of the software system, they are managed separately from the rest of C language program files in the project file organization. Meanwhile, includes.h and isr.h which correspond to these two documents will be placed individually in the root directory of header files.

In addition, header files and source files related to the overall framework process are placed in the sub-folders of Frame_H and Frame_C for categorized management. Frame_H includes four header files: type.h, MCF52233.h, GeneralFun.h and sysinit.h. type.h is used to define the alias for the type, it could simplify the keywords used for C language type definitions to a relatively short form. In this way, if the developer wants to define a variable, he needs not to type these long variable definition keywords. This also facilitates code reuse and transplantation between different compiling systems. MCF52233.h is the header file of Chip registers MCF52233 and associated bit definitions. It can be seen as the chip interface file, if there is no such file, we can not carry out any operation on the chip. sysinit.h corresponds to sysinit.c that is in the subfolders of Frame_C. It defines the basic parameters of the system initialization, such as the system clock, etc, while sysinit.c contains the actual initialization code. GeneralFun.h corresponds to GeneralFun.c, it provides a common and basic software functionality temper functions, such as the delay subroutine. These documents are indispensable for the normal operation of the system.

All above are necessary for normal operation of the system, they only form a "minimum system". If wanting the system to do practical things, one must also add the relevant function codes. In accordance with the component-based principles, these codes are placed in the subfolder of Component_H and Component_C. Each functional entity, or called component, corresponds to one.c file and one.h file. For example, "Light" component, for light control, corresponds to the Light.c and Light.h, which are placed in the sub-folders of Component_C and Component_H. Components are mainly divided according to the function. Besides "Light" component and "GPIO" component that defined in this paper, there is "serial communication", "Keyboard", "LED", "LCD" and other components, which are included in these two folders.

Finally, the embedded system engineering framework must include a special format file whose suffix name is. Lcf, which is an address linked file used to tell the compiler how the code is placed in a specific address space. Understanding the file format is conducive to a comprehensive understanding of the operation of embedded systems. Follow-up sections in this chapter will briefly describe its contents. Of course, the project documentation is also essential, it is very necessary for software developers.

SD-NOS engineering framework transplantation

Transplantation means a system uses components from other systems. For the same system as the core component, the portability is the largest among different application projects. Each project can use the same engineering framework as a template, design the head files and source files of low-level components according to the needs of each project, add them separately to “Frame_H” and include the file names of these head files in the “includes.h", then can quickly develop a new embedded application project. If the core component of the system changes, some header files of the low-level internal component and external functions, such as the module initialization function, will also change.Take the SCI component as an example, because the SCI module data register may be different, as well as the state register and its definition, settings involved in the initialization, such as baud rate, correspondence form, whether to verify and so on, are also different. So when the MCU changes, the related Macro in the header file and the implement code in the source of function SCI_Init also change. But the changes mentioned above is only a small part, moreover, the whole engineering framework is also unchanged. Consequently, SD-NOS is a scientific organization, a clear structure, reusable, portable engineering framework.

SD-NOS ENGINEERING FRAMEWORK APPLICATION

This section takes "high temperature furnace tempera-ture control system" as an example to introduce the design methods in the component-based embedded development.

Application system basic function

"High temperature furnace temperature control system" is mainly composed of the PC, hubs, and a number of temperature collectors. Temperature collectors collect the temperature on each point via the sensor and have a real-time display of them on the LCD screen. Press the "Start" button on the temperature acquisition device, then it will detect whether the deviation between the measured temperature (T) and the pre-set reference value (T0) in time (t0) has exceeded the allowed fluctuations ΔT, if has, then outputs the alarm signal. When having received data transmission request from PC, the temperature collector will send the temperature value measured to the PC through the network. The PC-side software can analyze the temperature, and draw the time - temperature curve. System components are as shown in Figure 3.

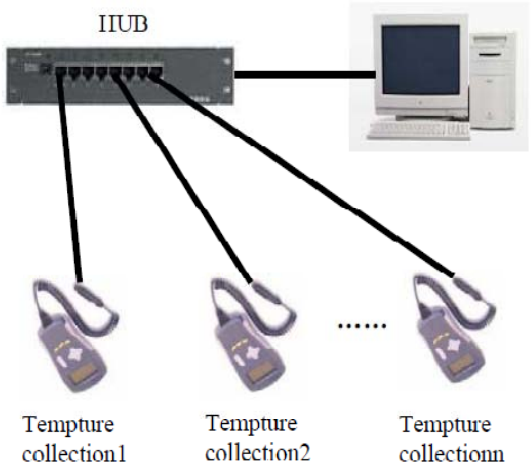

Figure 3: Construction of high-temperature furnace temperature control system

Hardware component design

Before carrying out the specific hardware design, the MCU must be selected in accordance with the function to be achieved. Taking into account that this system needs the temperature sensor to detect the temperature of the high-temperature furnace, and the network to transmit relevant data to the PC, ColdFire MCF52233 containing the A/D converter module and the network communication module is chosen as the system MCU.

After the MCU is determined, the power supply should be considered next. Due to the fact that MCF52233 MCU needs 3.3V power supply, and signal amplification circuits, relays, etc. need 5V power supply, while the external power supply is +24 V, so power modules are needed to convert the +24 V supply into + 5V and +3.3 V. The role of the signal amplification circuit is to amplify the input signal. After the MCU's A / D module converts the signal into digital signal and the filtering, physical quantity regression convert the digital into the actual temperature, the LCD displays the actual temperature. Because MCF52233 MCU's GPIO port has limited resources, LCD module may use the serial data line input mode. To facilitate the user's setting of some working parameters such as temperature reference value, the maximum deviation value allowed, etc. a number of keyboard keys can be provided. When the temperature fluctuation exceeds a certain range, the alarm signal will be output by the relay. Temperature information is sent to the PC via the network interface. In this case, the network signal line and the 24V power input signal line are bound together and connected to the outside world through the 9-pin serial port. In addition, in order to facilitate the test of the current system status, lights function as conflict detection indicator, network connection indicator, response indicator, running indicator, error indicator and so on can be designed.

Through the above analysis, according to component-based concept, "high tem-perature furnace temperature control system" contains the following hardware components[17]: MCF52233 minimum system, power components, LCD display components, Light (LED) components, temperature sensor signal amplification components, relay component, communication interface components, as well as the keyboard component. Their function and type are listed in TABLE II.

TABLE II: Hardware component partition of high temperature furnace temperature control system

Component name	Function	Type
M52233-MinSys	minimum system of MCF52233 MCU, contains BDM circuit	Core component
Power	24V Convert +24V to +5V and +3.3V	Intermediate component
LCD	Show data by serial input mode	Terminal component
Light	Turn on or turn off the light according to the output voltage	Terminal component
TemEnLarge	Amplify temperature sensor voltage, then output	Terminal component
Relay	whether relay is on or off is by the input voltage. The relay outputs	Intermediate component
NetPort	Use 9 pin SCI interface as net interface	Terminal component
KeyBoard	Output high voltage in defaul, press the key output the low voltage	Terminal component

Software component design

In this case the software is divided into two parts: low-end software design and high-end software design. The high-end software design uses high-level language C# development, due to space limitations, here is omitted. The low-end software design mainly introduces low-level components, which is a hardware driver and the cornerstone of building the low-end software. In low-end software design, the functions to be achieved by the software must be combined to design and plan the low-level components in accordance with component-based thought. This case involves the following low-level components:

(a) Universal IO (GPIO) components: initialize the port, set the port status, and access to the port status;

(b) Light (LED) components: light or extinguish error indicator, running lights and so on;

(c) Flash components: write the specified content into the designated Flash unit, set the system operating parameters and so on;

(d) Ethernet physical layer transceiver (EPHY) components: initialize EPHY module, access to the network connection status;

(e) Fast Ethernet Controller (FEC) components: initialize FEC module, send and receive Ethernet frames;

(f) Timer (GPT) components: Initialize the timer;

(g) analog-digital conversion (AD) components: initialize A / D converter module, start A / D conversion, the mean filtering;

(h) KeyBoard components: scan and obtain the value of the keyboard keys;

(i) Liquid crystal display (LCD) components: initialize LCD, display character data and so on.

SD-NOS engineering framework

In this case SD-NOS engineering framework structure organization is shown in Figure 4. “main.c” is main program file, and “isr.h” and“isr.c”are respectively interrupt handler header files and program files. Compared with Figure 2 “SD-NOS engineering frame-work for light project”, it is easy to see that the only difference between the two is that folders of “component_H” and “component_C” are added with Flash, EPHY, FEC, GPT, AD, KeyBoard, and LCD components header files and program files[18].

CONCLUSION

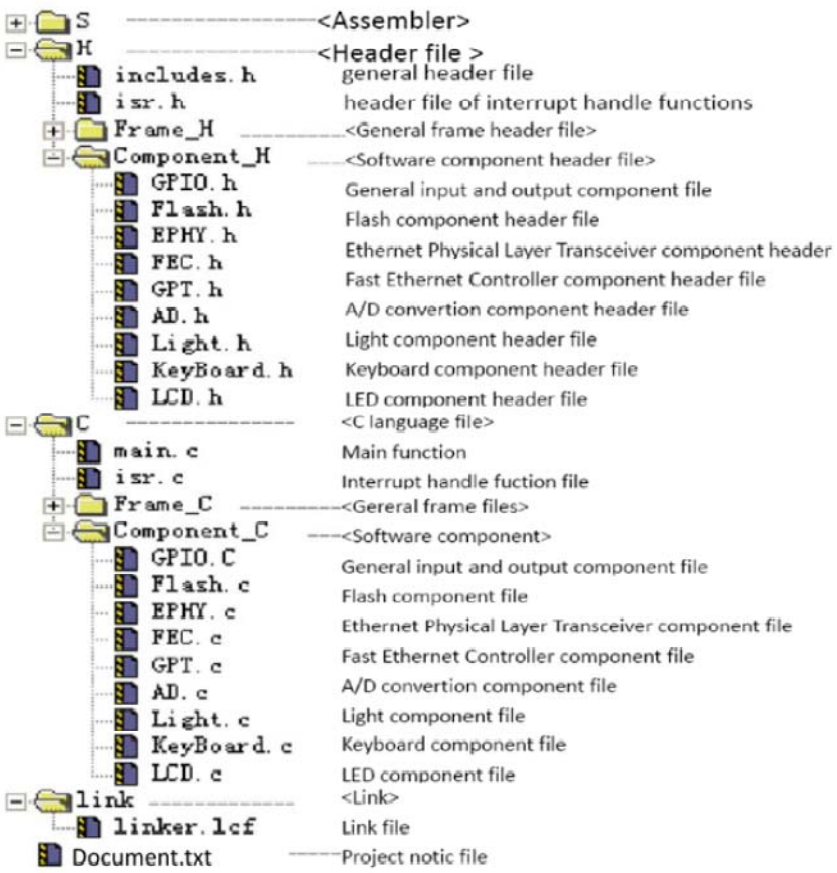

Figure 4: SD-NOS Engineering framework for high temperature furnace temperature control system project

Embedded system is a complex of software and hardware。With the progress of technology in recent years, hardware of embedded system has made great development in the performance, reliability and other fields; meanwhile, software development of embedded system has been unsatisfactory, especially in the MCU-based field without operating system. In view of this situation, the paper starts from the collaborative design thought of embedded hardware and software, adopts embedded software engineering principles and basic theory of component, details the definition and classification of embedded hardware components and software components and principles and methods of component design, and proposes the component-based engineering framework SD-NOS in the embedded software development. The paper analyses the engineering framework and its transplantation methods in detail, and provides the component-based Steps and methods of embedded development by means of a specific embedded project application under the SD-NOS framework. Currently, SD-NOS has been transplanted to S08, Kinetis K60, KL25 and other embedded development systems, the results of Tests and applications indicate that it is scientific, portability and high efficiency, and it can effectively reduce the technical difficulty and complexity of embedded development.

ACKNOWLEGEMENT

The project was supported by following funds: Priority Academic Program Development of Jiangsu Higher Education Institutions, Natural Science Foundation of Jiangsu Normal University (No.11XLA04, No.13XLA05)

REFERENCES

[1] R.K.Cavin, P.Lugli, V.V.Zhirnov; "Science and engineering beyond Moore's Law," Proceedings of the IEEE, **100**,1720-1749, May **(2012)**.

[2] I.Crnkovic; "Component-based software engineering for embedded systems," ICSE'05,712-713, March **(2005)**.

[3] F.Q.Yang, H.Mei, K. Q. Li; "Software reuse and software component technology," Acta Electronica Sinica, **27(2)**,68-75,51, February **(1999)**.

[4] M.Morisio, C.Tully, M.Ezran; "Diversity in reuse processes," IEEE Software, 56-63,July **(2000)**.

[5] F.A.Moreira, D.Nascimento, M.F S Oliveira; "A model-driven engineering framework for embedded systems design," Innovations in Systems and Software Engineering,**8(1)**, 19-33, January **(2012)**.

[6] W.H.Hu, W.Zhao, S.K.Zhang; "Study of application framework meta-model based on component technology," Journal of software, **15(1)**, 1-8, January **(2004)**.

[7] W.Emmerich, N.Kaveh; "Component technologies : Javabeans, Com, CORBA, RMI, EJB, and the CORBA component model, " *P*roceedings of the 24th international conference on software engineering, 691-692, March **(2002)**.

[8] G.Z Xiong, J.Y.Zhan; "Survey on techniques of SoC hardware/software co-design," *Computer Applications,* **26(4)**, April **(2006)**.

[9] L.F.Wang; "Component-based performance-sensitive real-time embedded software," *I*EEE Aerospace and Electronic Systems Magazine, 28-34, January **(2008)**.

[10] Bachman; "Technical concepts of componen-based soft-ware Engineering" CMU/SEI 2000-TR-8, May **(2000)**.

[11] Z.H.Gu, S.Kodase, S.Wang, K.G.Shin; "A model-based approach to system-level dependency and real-time analysis of embedded software," Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium *(RTAS'03),* 78-85, September **(2003)**.

[12] M.Lin, M.Rong, G. Q. Zhang; "Research on component-based compositional timing analysis for embedded real-time software," Computer Engineering and Applications, **45(11)**, 69-73, November **(2009)**.

[13] Z.T.Hu, Y.H.Wang; "Component-oriented general GPIO driver design of ColdFire series MCUs," *M*icrocomputer Information, **28(4)**, 69-71, April **(2007)**.

[14] P.A.Hsiung, S.W.Lin; "VERTAF:An Application Framework for the Design and Verification of Embedded Real-Time Software," IEEE Transactions on Software Engineering, **30(10)**, 656-674, October **(2004)**.

[15] Q.Wang, C.L.Du, L. Gang; "Study of general framework model for embedded real-time software," Computer Engineering and Design, **28(6)**,1372-1375, March **(2007)**.

[16] S.L Zhu, Y.H.Wang , D.W.Feng; "Research on MQX RTOS Component Engineering Framework, " Journal of Wuhan University of Technology, **35(10)**, 135-140, October **(2013)**.

[17] T.Sun, X.J.Yan, Y. Yan; "A chain-type wireless sensor network in greenhouse Agriculture, " Journal of computers, **8(9)**, 2366-2373, September **(2013)**.

[18] Y.H.Wang, J.M.Chen,Y.Z.Jiang; "Embedded systems designed and application with ColdFire", China: Pub-lishing House of Electronics Industry, **(2011)**.