Proposal for Training Embedded Software Engineers Using a Real-time Kernel Implementation Practice Program

Toshio Yoshida
Intelligent Sensing System Research Center
Toyohashi University of Technology
1-1 Hibarigaoka Tenpakuchou, Toyohashi-city
Japan
yoshida@gcoe.tut.ac.jp

Jun Sawamoto
Faculty of Software and Information Science
Iwate Prefectural University
152-52 Sugo, Takizawa, Iwate 020-0193
Japan
sawamoto@iwate-pu.ac.jp

Abstract: One of the important issues recognized recently is how to educate embedded-software engineers. With embedded systems, the real-time kernel plays an important role in basic controlling. This paper proposes an implementation practice program for embedded software engineers to be trained so that each is able to write the program code of a real-time kernel and check to see how it can function to support the design goals. Described herein also are the new real-time kernel, μK, that was specifically developed for the training use, and the results of having applied it for the implementation practice training program with μK at the “School of Monodzukuri (Manufacturing) 2008 for embedded system” (as organized by the Embedded Technology Center at Iwate Prefectural University).

Key-words: Embedded system, Embedded software, Real-time kernel, Skill standard

1 Introduction

One important issue discussed recently is how to train embedded-software (SW hereinafter) engineers. Especially important is the issue of how to train highly skilled engineers to be capable of system design (1). The IPA/SEC defines the framework of the embedded skill standard as given in the Embedded Technology Skill Standards (ETSS) (2), in which the skills necessary for embedded-SW engineers are classified into two categories of so-called “discipline skills for special fields” and “common skills to all fields”. The former are skills associated directly with the functionality that is necessary to make any specific product workable, which include robot motion-control technology, and wireless network technology for cell phones, for example. The “common skills to all fields” are those that are not dependent on the field of any specific product.

On the other hand, any embedded-SW is organized with the three-layered SW group to put into the associated embedded hardware including the real-time kernel, the middleware, and the application SW associated (3)(4). The real-time kernel (RTK hereinafter) in the lowest layer of all manages execution control over the application tasks and access controls to the hardware.

Understanding the RTK is unavoidable to know how to design the embedded system and how it must function (5).

When developing a product incorporating an embedded system, an RTK that is already available in the marketplace is used in most cases. However, a

<table>
<thead>
<tr>
<th>Embedded software common skill</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements of Technology</td>
</tr>
<tr>
<td>(1) MPU</td>
</tr>
<tr>
<td>(2) Basic I/O systems</td>
</tr>
<tr>
<td>(3) System design</td>
</tr>
<tr>
<td>• Task coordination framework</td>
</tr>
<tr>
<td>• Data structure design</td>
</tr>
<tr>
<td>(4) Middleware</td>
</tr>
<tr>
<td>• File system</td>
</tr>
<tr>
<td>• Graphic User Interface</td>
</tr>
<tr>
<td>• TCP/IP protocol stack</td>
</tr>
<tr>
<td>(5) Real-Time Kernel</td>
</tr>
<tr>
<td>Software Development Methodology</td>
</tr>
<tr>
<td>(6) Requirement analysis and Specification description</td>
</tr>
<tr>
<td>(7) Program design</td>
</tr>
<tr>
<td>(8) Language (C, Assembler)</td>
</tr>
<tr>
<td>(9) System programming</td>
</tr>
<tr>
<td>(10) Software testing</td>
</tr>
<tr>
<td>(11) Software development tools</td>
</tr>
<tr>
<td>Management and Control</td>
</tr>
<tr>
<td>(12) Project management</td>
</tr>
<tr>
<td>(13) Developing process Control</td>
</tr>
</tbody>
</table>
2. Required Skills for Being an Embedded-SW Engineer and Training Program

## Embedded-SW Engineer and Training Program

### 2.1 RTK and “common skills to all fields”

As presented in Table 1, the ETSS is classified into “common skills to all fields” to be required to build any embedded-SW into three categories: technical elements, SW development techniques, and management skills (2). The RTK building technique is therefore positioned as a core of the technical elements of the “common skills to all fields” of the ETSS, which means it is a necessary technical element that any embedded-SW engineer must have.

### <2.2> Embedded-SW engineer training in RTK implementation practice: To implement an RTK, what is required also is to understand the development languages to describe how RTK is processed, the system programming technique, the SW development environment, the CPU execution behavior, and the associated hardware interface in addition to understanding its specifications. Comparing each “common skill to all fields” defined in the ETSS, as shown in Table 1, with the skills required to implement an RTK as described above, it is apparent that (1), (2), and (5) of the technical elements of the Table correspond to (8), (9), and (11) of the SW development techniques, respectively. Therefore, the skills necessary to implement an RTK correspond to many items of the technical elements and the SW development techniques of the “common skills to all fields” for embedded-SW development. Consequently, the RTK implementation practice program proposed herein is a useful method to have a trainee acquire “common skills to all fields” that are necessary for embedded-SW engineers.

### 3. Real-time Kernel, µK, and a Proposal for Implementation Practice Program

#### 3.1 Proposal for Real-time kernel, µK

- **3.1.1 Functionality and specification of µK**

To understand how an embedded system works, one must know how each of the tasks that function the associated application SW would work, as well as of the cooperation mechanism of tasks. For our µK, we have the task control functionality, which must be implemented as the main functionality of all, getting rid of memory control and the message communication capability that a common real-time kernel has. In fact, µK implements eight system calls that are categorized into the SVC-1 group and the SVC-2 group.

<table>
<thead>
<tr>
<th>µK</th>
<th>µK1</th>
<th>µK2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) H/W initialization</td>
<td>(6) Kernel data of initialization</td>
<td>(9) Kernel data of SVC-2 group initilization</td>
</tr>
<tr>
<td>(2) Save context</td>
<td>(7) Task scheduler</td>
<td>(10) System calls of SVC-2 group</td>
</tr>
<tr>
<td>(3) Infinite loop</td>
<td>(8) System calls of SVC-1 group</td>
<td>- SVC_pause(), SVC_resumeTASK()</td>
</tr>
<tr>
<td>(4) IT handling</td>
<td></td>
<td>- SVC_takeSEMA(), SVC_giveSEMA()</td>
</tr>
<tr>
<td>(5) Restore context</td>
<td></td>
<td>- SVC_startTASK()(), SVC_exit()</td>
</tr>
</tbody>
</table>

#### 2.2 Structure of µK0, µK1, µK2

![Fig.1. Structure of µK0, µK1, µK2](image)

As described in this paper, an implementation practice program of RTK is proposed using the specially and newly developed RTK (called µK hereinafter) to train embedded-SW engineers. Additionally described herein is such an implementation practice program used for the “School of Monodukuri 2008 for embedded system”, as organized by the Embedded Technology Center at Iwate Prefectural University, with information related to how it was responded.

### 2
confirming the step-by-step level of understanding that we can proceed with the training program while for each level that a trainee needs. Another benefit is to provide the right implementation practice program step-by-step level of specifications, it becomes possible μ three steps:

1. **First step: μK0**
   - Where the CPU is initialized, and the basic loop and interrupt processing portions are implemented.

2. **2nd step: μK1**
   - Where, in addition to the μK0 processing, the task scheduler and two system calls of the SVC-1 group (SVC_startTASK, and SVC_exit) are implemented.

3. **3rd step: μK2**
   - Where six more system calls of the SVC-2 group are added to the μK1, making eight system calls added to μK1.

   The μK0 is a single task system having only a basic loop and an interrupt processing in it. The μK1 has a multi-task capability, and the μK2 has more system calls added to μK1.

3.1.3 **Considerations to be as RTK for training use**

1. **Three steps of specification**
   - For use as a step-up type of RTK implementation practice training, we divide the implementation of μK into three steps—μK0, μK1, and μK2—to provide the proper implementation practice program conforming to the trainee’s level and the period during which the trainee learns it. The μK1 includes the specification of μK0, and the μK2 the specification of μK0 and μK1, respectively.

   (1) **First step: μK0**
   - Where the CPU is initialized, and the basic loop and interrupt processing portions are implemented.

   (2) **2nd step: μK1**
   - Where, in addition to the μK0 processing, the task scheduler and two system calls of the SVC-1 group (SVC_startTASK, and SVC_exit) are implemented.

   (3) **3rd step: μK2**
   - Where six more system calls of the SVC-2 group are added to the μK1, making eight system calls implemented.

   The μK0 is a single task system having only a basic loop and an interrupt processing in it. The μK1 has a multi-task capability, and the μK2 has more system calls added to μK1.

3.2 **Implementation practice program of μK**

3.2.1 **The implementation practice program objective**

   The goal that must be accomplished by the implementation practice program is that the program trainee be able to understand the following six “common skills to all fields” necessary for embedded-SW engineers.

   (1) Understanding RTK: Understanding its basic functionalities and specifications
   (2) Development language: Understanding C and Assembler languages and associated others
   (3) System programming technique
   (4) Operations in the development environment
   (5) Understanding CPU execution behavior
   (6) Understanding hardware interface associated

3.2.2 **Information on implementation practice programs**

1. **Implementation practice program outline**

   Figure 2 shows the sequence in which the implementation practice program of μK is given to trainees. An introductory lecture to RTK is provided first to explain the basic RTK concept. Then it is followed by practical training on how to operate in the given program development environment. After these preparations made as described above, we move to the actual implementation practice of μK. It starts with the μK0 followed by μK1 and μK2 in that sequence. Each of the μK0, μK1, and μK2 is separated into further smaller step of programs for which the implementation practice training is carried on. The training period lasts 5 days (40 hr), the trainees are restricted to 20 or fewer, with at least three instructors for lectures and practice training support.

2. **Textbooks and facilities used**
Following are the textbooks and the facilities used for this training. Each trainee is presumed to use each set of both the textbooks and the facilities.

Facilities used
- CPU board (loaded with 16-bit CPU) (7)
- SW development tools (HEW)(9)
- Personal Computer (PC)

Textbooks
- Introduction to Real-time Kernel (written in PowerPoint (called PPT hereinafter) )
- Specification of µK, and description for implementation (in PPT)
- Textbook for µK implementation practice

The textbook for µK implementation practice training listed in the third position above is used for training on the actual µK implementation practice. It is the textbook with which an implementation practice proceeds and which contains descriptions of how to use the associated SW development tools followed by the necessary implementation steps up to the µK1 completion, with about 30 programs provided, each of which is described in 10 lines of code or so.

4. Execution of Implementation Practice Program of µK and Evaluation

4.1 Training seminar and how it was conducted

The implementation practice program of µK were conducted as one course (9) of “School of Monodzukuri 2008 for embedded system” (organized by the Embedded Technology Center at Iwate Prefectural University). The following describes how it proceeded.

(1) Program trainees

The seminar was organized by 13 students of the undergraduate and the master’s course of the university and 5 young engineers from the software houses: 18 people. The knowledge of RTK by those trainees was not deep but rather a shallow level. None of them knew of RTK in detail.

(2) Training period and hours spent

The implementation practice program of µK lasted 5 days, with 8 hr condensed seminars given every day.

(3) Scope and schedule of the implementation practice of µK

No trainee had experience of programming or real-time kernel transplant. Because there were five days available in all, the objective for this training program was planned up to the actual implementation of µK1.

(4) Implementation practice proceeding

Training progressed using the “textbook for µK implementation practice” and checking on how every step of the implementation was processed. The trainee carried on with the code-and-build (compile and link) operations for every programming step, downloading their program into the CPU board, executing it, checking the memory information changes, if any, as well as the registers when a break point occurred, and making the associated LED light of the CPU board flicker to verify whether the program was working properly. The instructors, after checking the progress of each trainee, showed each associated program by code as a sample. Consequently, they viewed the manner in which the seminar proceeded.

4.2 Results of implementation practice program execution

Up to the second day of the seminar for training, the trainees—aside from one person who had good experience in programming—did not spend much time for their own program coding, but instead merely concentrated on copying the sample codes that the instructor provided. However, on the third day and thereafter, almost all trainees understood the source code of the µK. Then they, except the three people who did not know the C language sufficiently well, tried coding their own program. Table 2 shows the step-by-step achievement made on the implementation practice program by each of the 16 trainees who responded to the questionnaire given to them. The number shown in each of the grid space stands for the number of people associated. Before the seminar started, we had expected that just about a half of the trainees would be able to verify how the µK1 would behave. Results show that all trainees except one were able to do it. One trainee who had majored in computer languages and who had good experience of the C language was able to finish all tasks up to the implementation of the µK2.

5. Evaluation and Considerations

Results show that, using the step-up type of RTK as a training material and using the step-by-step implementation practice program we administered to trainees for 40 hr or so, the trainees who had had practically no knowledge of RTK were able to code an RTK program so that they could verify the behaviors of RTK and its associated tasks. Furthermore, we ensured that they could learn the “common skills to all fields” of the ETSS in an efficient way through the implementation practice program of the µK.

5.1 Evaluation and issues on µK

5.1.1 Evaluation on µK

The task scheduling process is the core part of task control. That the trainees fully understand this part is the key verification that they understand RTK. In µK, we used the scheduling process of ready-state task scan method, which is easy for trainees to understand as described in <3.1.3>. By that, no trainee had much trouble with coding the scheduler part. In general, an RTK would often perform its scheduling process based on the waiting queue of the ready-made tasks. Consequently, the program code would become more complex than without it; we thought it would deter the trainee from the practice training. For that reason, we used no waiting-queue method in our µK.

5.1.2 Issues of µK and how to resolve them

Because
the RTK available in general has functionality of many kinds, we tend to try to have as many types of functionality as possible in the RTK for training use as well. It is important that we specifically examine the minimum level of specification to have for training use, which means that it must be compact, easy to understand and easy to implement for the trainee. It might be effective if we had the specification of intermediate level between μK1 and μK2, for example, adding to the μK1 only two system calls, SVC_pause and SVC_resumeTASK, obtained from the SVC-2 group, because the μK2 level might be too large of a specification for training use.

5.2 Evaluation and issues related to implementation practice program

5.2.1 Evaluation of implementation practice program

The implementation practice program introduced and proposed herein is extremely beneficial for use. We can say that when examining the achievement level that every trainee accomplished the tasks and from the replies to the questionnaire given to them. Especially, by making the specification of RTK as simple as to be able to use for the introductory training, and making the scope of implementation practice smaller, the trainee was able to understand it easily. Furthermore, we confirmed that we had practically no dropouts for the implementation practice program. It was accomplished by the fact that we divided the program implementation into smaller steps of the program and gave lectures that way repeatedly.

Table 3 presents resultant points with respect to the level of skill acquisition for each item of the implementation practice program. The skill acquiring level values (titled “Results”) are the averaged points that all trainees received in terms of how much the trainee acquired skills to the level that each targeted to understand each item of the implementation practice program. When the trainee reports acquiring a level of understanding that is as high as the implementation practice program, μK1, its value is 100. It is 80 when the understanding was achieved mostly as expected, 60 when it was as much as expected, and 50 when it was an entirely unexpected level. A judgment was made in terms of the trainee’s level of understanding for each sample of the program code that the trainee wrote, the execution state of the sample task on the CPU board, the reply status to the questionnaire given, and other information (that includes the operation status in the development environment given and the questions made by the trainees). Then every associated point was weighted for each skill-judging item and added up by each skill acquisition item to produce the final point value, as presented in the table. The resultant points for “understanding of RTK”, “understanding of development environment”, “understanding of CPU execution behavior” and “system programming technique” were all just about equal to the expected values.

Because we restricted the scope of the implementation practice to the level of specification, as for μK1, and proceeded with the lectures particularly addressing the task control feature, which is the core part of an RTK, we believe that it must have been easy for the trainee to understand μK and to implement it as a result. In addition, we believe that the trainee was able to understand the “RTK specification” and the “system programming technique” smoothly because we had a series of lectures in the way that we released the associated and well-prepared small programs a little by little.

We obtained low-skill acquisition levels of 69 and 67 points, respectively, for the two items, “understanding of the development language” and the “hardware interface”. During the course of the μK implementation, each trainee improved their own programming skill in the C language, but not to the level that was expected in the beginning. That was true because the kernel data structure of the μK was given to the trainees from the instructor side with not much consideration taken, although the skill level of each trainee and the progress of the implementation practice program were examined. Consequently, the trainees had few chances to consider the kernel data structure for themselves. Another reason was that the implementation practice scope was restricted to μK0 and μK1; therefore, the trainees had no opportunity to learn of programming on the queued data structure, which is necessary for μK2 implementation.

5.2.2 Issues of practice program implementation and resolution

(1) Understanding the programming language

The difficulty that the trainee might have during the implementation of practice training course is not a lack of understanding the RTK specifications, but lack of knowledge necessary to code a program in the C language. For those trainees with little experience in programming in the C language, a way to have them read a few hundred lines of code, for example, before they receive the training course would be highly advised. That small amount of knowledge would give them much better understanding.

(2) Giving self-learning opportunity

To help the trainee acquire programming skills, it would be very effective for the trainee to avoid copying the sample program that the instructor provides, and
instead to design the program and write the code of the program independently. During the short time period of the implementation practice program provided to the trainees, it would be difficult to give ample time to the trainees so that they can think about the program that every one of them must write. It is important at the very least that we restrict the lectures to necessary matters, and give them sufficient time for that.

(3) More time to give task program implementation practice

Having each trainee debug every task would help trainees to understand the task behavior, which then would lead them to understanding of the RTK. Therefore, it is an important factor during the training course of the task program implementation practice. Providing trainees with more time for task programming when the time of the entire training session is allocated would be the most efficient way that a training program can accomplish this.

6. Conclusion

We first developed the step-up type of RTK called μK; using it as a training material, we conducted the RTK implementation practice training program for college students and SW engineers. Through that training program, we ensured that the RTK implementation practice program we developed was an efficient way for the trainee to acquire the “common skills to all fields” of the ETSS for embedded-SW engineers.

Being an embedded-SW engineer and creating an associated product requires skills for the specific field of how to construct a piece of application SW, for example, in addition to acquiring the “common skills to all fields”. How to have a trainee acquire such skills efficiently is the issue that we would have to consider and incorporate into our training program in the future.

References:

---

### Table 3. Acquisition of Embedded SW Skills by μK Implementation Practice Program

<table>
<thead>
<tr>
<th>Items for understanding</th>
<th>Judging points</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Program code by students</td>
<td>Execution on CPU board</td>
</tr>
<tr>
<td>1 Specification of μK</td>
<td>27/30</td>
<td>28/30</td>
</tr>
<tr>
<td>2 Development languages (Assembler, C)</td>
<td>35/50</td>
<td>28/30</td>
</tr>
<tr>
<td>3 System programming method</td>
<td>38/50</td>
<td>28/30</td>
</tr>
<tr>
<td>4 Software development tools</td>
<td>8/10</td>
<td>18/20</td>
</tr>
<tr>
<td>5 Behavior of CPU execution</td>
<td>32/40</td>
<td>38/40</td>
</tr>
<tr>
<td>6 Hardware(H/W) interface</td>
<td>16/20</td>
<td>19/20</td>
</tr>
</tbody>
</table>