

引用格式:孙亚军,王林,王宜怀,等. 基于 GEC 架构的 mbedOS 驻留技术研究[J]. 微电子学与计算机, 2020, 37(6): 12-18.
[SUN Y J, WANG L, WANG Y H, et al. Research on mbedOS resident technology based on GEC architecture[J]. Microelec-
tronics & Computer, 2020, 37(6): 12-18.]

基于 GEC 架构的 mbedOS 驻留技术研究

孙亚军¹, 王 林¹, 王宜怀¹, 刘长勇², 程宏玉¹

(1 苏州大学 计算机科学与技术学院, 江苏 苏州 215006;

2 武夷学院 认知计算与智能信息处理福建省高校重点实验室, 福建 武夷山 354300)

摘 要: 针对当前 MCU 在实时操作系统上进行嵌入式应用程序开发的需求, 提出使用 GEC 架构, 并且在该架构下实现 mbedOS 操作系统驻留内存, 其中提出合理划分 Flash 和 RAM 空间设计规则, 总结出使用 API 动态链接技术实现用户程序调用接口程序进行编程, 实现嵌入式编程和通用计算机编程类似, 缩短编译时间, 加快开发效率。最后分别在 S32K144 和 MKL36Z64 进行设计实验, 实验表明该技术具有可行性。

关键词: GEC 架构; mbedOS; 操作系统驻留; Flash 和 RAM 空间设计; API 动态链接

中图分类号: TN791

文献标识码: A

文章编号: 1000-7180(2020)06-0012-07

Research on mbedOS resident technology based on GEC architecture

SUN Ya-jun¹, WANG Lin¹, WANG Yi-huai¹, Liu CHANG-yong², CHENG Hong-yu¹

(1 School of Computer Science and Technology, Soochow University, Suzhou 215006, China;

2 Key Laboratory of Cognitive Computing and Intelligent Information Processing of
Fujian Education Institutions, Wuyishan 354300, China)

Abstract: In view of the current MCU requirements for embedded application development on real-time operating systems, it is proposed to use the GEC architecture, and implement the mbedOS operating system resident memory under the architecture, which proposes a reasonable division of Flash and RAM space design rules, and concludes The API dynamic link technology is used to implement the user program call interface program for programming, which realizes similar embedded programming and general computer programming, shortening the compile time and speeding up the development efficiency. Finally, the design experiments were carried out in S32K144 and MKL36Z64 respectively. The experiment shows that the technology is feasible.

Key words: GEC architecture; mbedOS; Operating system reside; Flash and RAM space design; API dynamic link

1 引言

近年来,微控制器(Micro Control Unit,MCU)在计算性能和内存容量巨大提升,嵌入式应用程序开发和通用计算机程序开发越来越相近,为此提出类似于通用计算机开发的嵌入式架构——通用嵌入式计算机(Generate embedded computer,GEC)架构,该架构将具体工程分为基本输入输出系统程序(Basic Input and

Output System,BIOS)与用户程序(USER)两个部分^[1]。并且将 mbedOS 驻留于内存中,使得嵌入式应用程序开发和在通用 PC 上开发类似,通过直接调用应用程序编程接口(Application Programming Interface,API),实现应用程序的开发,降低开发门槛,同时还可以减少程序编译时间,提高开发速度。针对嵌入式操作系统(Embedded Operate System, EOS)移植的文献丰富,如文献[2][3]主要实现 uC/OS-II 在 Arm9 和 Arm7 上的移

植,文献[4]实现了 MQX 在移植与应用,文献[5]则是对 FreeRTOS 操作系统在 S3C44B0 处理器上的移植与实现,文献[6]实现了 uC/OS-III 的移植与实现等,而对于嵌入式实时系统驻留不多.文献[7]虽提到操作系统的驻留,但都是针对 PC 机,对于微控制器或者其他嵌入式开发并没有涉及. mbedOS 操作系统驻留技术难点在于 Flash 和 RAM 分区设置以及 API 表链接技术.文章针对以上问题设计出合理方案,最后分别在 S32K144 和 MKL36Z64 微控制器上实现 mbedOS 操作系统的驻留,并且设计相应的实验验证其可行性.

2 GEC 架构与 mbedOS 操作系统概述

mbedOS 是 ARM 公司在 2014 年针对 Cortex-M 及 Cortex-A 系列推出的嵌入式物联网实时操作系统^[8,9]. mbedOS 提供了许多系统服务,如系统时间相关操作函数,事件机制、消息队列机制、信号量机制、邮箱机制等^[10,11].

GEC 架构主要包括 BIOS 程序模块与 USER 程序模块,因此用户只需针对 USER 编程,而无需了解底层寄存器的配置、驱动编写以及操作系统实现细节等,提高了程序的可移植性和编程颗粒度,使得开发变得更加容易,其结构如图 1 所示.

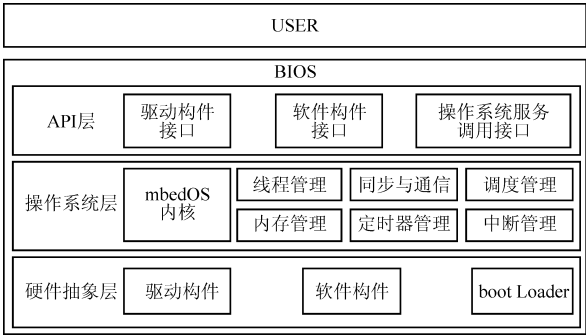


图 1 GEC 架构图

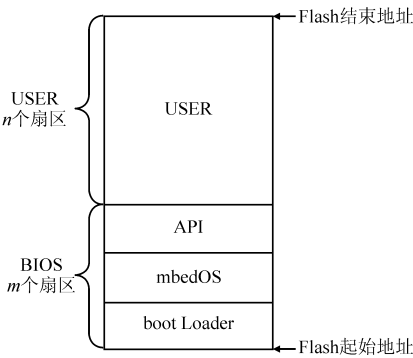


图 2 Flash 分区设计

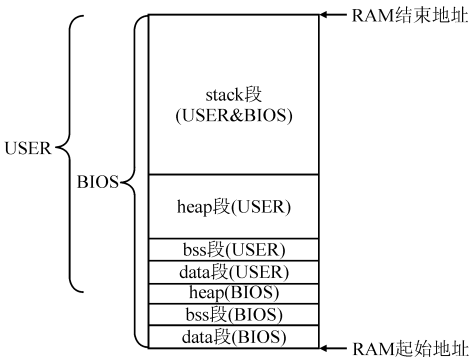


图 3 RAM 共享栈空间方式

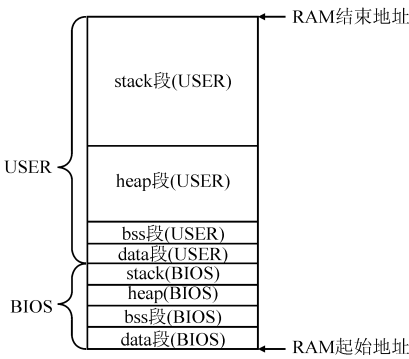


图 4 RAM 独享栈空间方式

2.1 BIOS 程序工程框架

从图 1 可以看出 BIOS 程序模块主要包括硬件抽象层、mbedOS 操作系统层以及 API 层. 其中硬件抽象层主要包括驱动构件、软件构件以及硬件初始化 bootLoader 程序. 操作系统层则是由 mbedOS 内核和线程、调度、内存等管理程序组成. API 层为 BIOS 向 USER 提供接口服务,包括驱动接口、软件构件接口和操作系统构件接口. BIOS 程序模块为驻留在芯片中的程序,是芯片上电后首先被执行的一段程序,主要负责系统初始化、mbedOS 驻留以及 API 表初始化. 为了更好地组织 BIOS 程序模块,遵循嵌入式软件工程的基本理论,按照嵌入式软件构件设计基本思想和基本原则,设计 BIOS 工程框架如表 1 所示,体现按照“分门别类、各有归处”原则.

2.2 USER 工程框架

USER 程序是用户根据自己需求自行开发的程序,它可以调用 BIOS 中 API 层提供的服务,实现对硬件驱动构件、软件构件、操作系统服务的调用等,而无需再次进行底层驱动构件开发,降低了两层间的耦合性. 同时,USER 程序只需要修改少量的配置信息,就可以实现在 GEC 架构下不同内核、不同 MCU 之间的移植,提高了软件的可复用性、可移植

性. USER 工程结构和 BIOS 工程框架类似,唯一区别在于 08 文件夹, BIOS 的 08 文件夹为 mbedOS 组件文件, USER 中为 08_RtosPrg, 用于存放任务线程.

表 1 BIOS 工程框架

目录结构	名称	存放文件	所在层
01_Doc	文档	工程说明文档	
02_Core	内核相关	内核相关文件,如 core_cm4. h 等	
03_MCU	MCU 相关	存放芯片链接文件、启动文件和底层驱动构件.	
04_GEC	GEC 相关	内含 gec. h 文件给出了 GEC 芯片引脚宏定义,可供用户编程硬件设计与编程使用	硬件抽象层
05_UserBoard	用户板	用户利用 GEC 做成用户板,封装成构件函数	
06_SoftComponent	软件构件	软件程序,与硬件无关,封装成软件构件,如字符串与整数转换	
07_NosPrg	无操作系统程序	放置总头文件(includes. h)、中断服务例程源程序文件(isr. c)、主程序文件(main. c)	
08_b=mbedOS	操作系统程序	包括 mbedOS 启动相关文件以及 mbedOS 封装的组件,如事件(EventFlags)、消息队列(Queue)、信号量(Semaphore)、互斥量(Mutex)等	操作系统层

3 GEC 架构下 mbedOS 驻留方法

要实现 mbedOS 的驻留,首先需要对 Flash 和 RAM 空间进行合理划分,其次需要设计出符合软件工程可移植、可复用、可拓展 API 层接口函数.

3.1 Flash 和 RAM 空间划分

在嵌入式开发中,中断向量、代码以及常量通常保存在 Flash 中,全局变量、静态变量以及局部变量存放在 RAM 中. 由于 Flash 和 RAM 存储空间资源有限,在 GEC 架构下设计 BIOS 和 USER 时,必须充分考虑 Flash 和 RAM 的划分,其划分可遵循如下原则:

(1)对 Flash 的划分采用互不干扰原则. 由于在 GEC 架构中程序分为 BIOS 和 USER 两个程序模块,但都存储在 Flash 中. 因此,必须对 Flash 进行合理划分,使得 BIOS 和 USER 程序互不干扰,其划分如图 3 所示, BIOS 占用 m 个扇区, USER 占用后 n 个扇区. 在实际的应用开发中,要根据具体的 MCU 的 Flash 大小合理分配 BIOS 和 USER 所占扇区大小,一方面必须保证 BIOS 程序能正常运行,另一方面 BIOS 所占扇区不宜过大,否则会造成 Flash 的浪费,导致 USER 程序所占 Flash 空间不足,只能以牺牲 USER 程序功能来满足程序需要.

(2)对 RAM 的划分采用最大化利用原则. 当 MCU 的 RAM 空间较小的时,可以采用共享栈空间方式,如图 4 所示;而对于 RAM 空间较大的情况则可以考虑采用独享内存方式,如图 5 所示. 在图 4 中, BIOS 占用整个 RAM 空间,包括 bss 段、data 段和 heap 段, USER 必须从 BIOS 的 heap 段之后开始使用 RAM,也包括自己的 bss 段、data 段和 heap 段, BIOS 和 USER 有共同的栈底地址(即 RAM 结

束地址+1). 在图 5 中,是将 RAM 划分为两个相互不重叠的区域,供 BIOS 和 USER 分别使用. 两种方式各有益处,共享栈空间方式可以最大化的利用 RAM 空间,不会出现过大的空间浪费,但是需要注意内存冲突,在 RAM 空间较小,如 16Kb 及以下时,推荐可以使用该方法. 独享内存方式实现简单,不会出现内存冲突,但是 RAM 空间利用率较低,通常在 RAM 空间较充足时使用,如 RAM 的空间大小为 60Kb 及以上时.

3.2 API 表动态链接设计

USER 程序若要对 BIOS 中函数的调用,就必须获取到被调用函数的入口地址,而 BIOS 程序和 USER 程序是两个相互独立的工程,因此 USER 程序不可以直接调用 BIOS 中的函数. 为此,在 GEC 架构下, BIOS 将能够提供给 USER 程序使用的函数进行统一设计、封装以及注册,将这些可被调用的函数入口地址有序地存放在 API 表中, USER 程序就可以通过访问这个 API 表实现对接口函数的调用. API 表的设计主要包括接口函数定义、声明与注册.

接口函数设计主要有两种方式,其一将接口函数表存放于固定的 Flash 内存中,在成功跳转至 USER 程序后, USER 程序通过直接读取相应地址获取接口函数的入口地址后进行重映射,该接口函数设计称为静态链接. 其二为动态链接,设计的 API 表不需要存放于指定位置 Flash 中, USER 程序通过 SVC 中断服务调用获取 BIOS 中的 API 表首地址后进行接口函数映射. 静态链接方法设计简单、易于实现,但是影响 Flash 内存,容易产生内存碎片,降低内存使用率. 动态链接实现较难,需要借助系统服务调用(Supervisor Call, SVC),但无需考虑

Flash 存放位置,最大化利用内存空间,不会产生内存碎片.

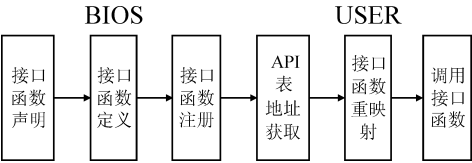


图 5 API 表设计与调用

3.2.1 接口函数定义与声明

接口函数主要是 BIOS 提供给 USER 程序使用的函数,其定义需要按照嵌入式软件构件化技术的基本原则,以构件为基础,包含源程序(.c 或 .cpp)和头文件(.h),源文件和头文件成对出现,文件名相同,后缀名不同. API 表并不囊括 BIOS 中所有的函数,主要包括硬件驱动构件、软件构件和操作系统服务调用构件三大类.

(1)硬件驱动构件

硬件驱动构件主要包括对硬件资源调用函数的封装,用户可以调用该类型构件进行硬件资源访问

与使用. 如控制一盏小灯的开关,可以通过调用 gpio 相关函数来控制对应引脚输出的高低电平来实现.

(2)软件构件

软件构件则是完全脱离硬件基础,将常用的一些算法或者字符串操作等函数进行封装. 如若想要实现字符串转整形,则可以调用 stringToInt 函数实现,方便用户开发.

(3)操作系统服务调用

mbedOS 源程序使用 C++ 编写,提供各种类,如线程类、消息队列类、互斥量类等. 为了方便使用,将 mbedOS 封装成构件,通过直接调用函数就可以实现操作系统服务调用. 如 thread_create 函数用于线程的创建其内部使用全局链表 list 或者数组存储这些创建的线程对象,创建成功后会返回相应的线程序号,通过线程序号可以对该线程进行线程相关的操作,如线程等待、线程设置信号量、线程销毁等.

表 2 中列出部分接口函数.

表 2 部分接口函数一览表

构件类别	接口函数名	函数参数	函数功能
硬件驱动构件	gpio_set	uint_16 port_pin, uint_8 state	输出电平信号设置
	uart_init	uint_8 uartNo, uint_32 baud_rate	串口初始化
软件构件	IntConvertToStr	int_32 num,uint_8 * buf	整形数转换成字符串
	ArrayCopy	uint_8 * dest,uint_8 * source,uint_16 len	内存拷贝
操作系统构件	OS_start	void (* func)(void)	操作系统启动函数
	thread_create	uint_32 priority,uint_32 stack_size	线程创建

3.2.2 接口函数注册

接口函数在完成定义和声明后,还需要进行注册. 接口函数的注册主要是完成将提供给 USER 程序调用的函数按顺序存放在 API 表中. API 表类似于一维数组,采用汇编语言编写,可以参考中断量表的形式进行设计,表元素为各接口函数入口地址,这样的设计有利于增强 API 表的可移植性、健壮性、可拓展性. 在汇编文件中其一般格式如下:

格式:

接口函数表名:

接口函数类型接口函数个数

接口函数类型接口函数名

举例:

BiosUserFun:

```
.long 104 //接口函数个数
.long sysTimeGet //系统时间获取
.long reserved_func//预留函数
```

其中函数类型是无符号长整型(即. long 类型),而不是指函数的返回类型. API 表的第一个元素存放的是 BIOS 提供给 USER 使用的接口函数总数,其后是有序排放的接口函数名. 为了方便 API 表的更新与扩充,在设计上使用缺省函数名来预留一些位置,在需要使用的时候,只需要进行替换就可以实现.

3.3 USER 接口函数获取与重映射

USER 程序能够正常调用接口函数,需要经过以下两个步骤:

(1)USER 获取 API 表首地址

BIOS 程序跳转到 USER 用户程序后,首先会触发 SVC 中断,转而去调用 BIOS 中的 SVC 中断服务程序,实现 BIOS 中 API 层接口函数表的首地址获取,并且将获取的函数表首地址保存在一个全局数组名变量中.

(2)USER 接口函数重映射

在获取 API 接口函数表的首地址后,并不能直接进行 API 函数调用,因为其中的表元素只是函数入口地址,并没有函数实现的具体形式,即不知道函数的具体形式,就需要进行函数地址的重映射,注意映射的顺序要和注册顺序一致.其一般形式如下:

格式:

#define 函数名 ((接口函数声明指针表达式)(全局数组[接口函数序号]))

举例:

#define gpio_get ((uint_8 (*)(uint_16 port_pin))(bios_fun_point[25]))

3.4 驻留的注意事项

(1)mbedOS 系统服务调用问题. mbedOS 的运行依赖于 SysTick(系统时间滴答)、SVC(系统服务调用)和 PendSV(可挂起服务调用)中断,而在 BIOS 和 USER 中使用到 SVC,因此在启动 mbedOS 前,应该确保 SysTick、SVC 和 PendSV 中断使用权交给 mbedOS,其具体实现可以通过中断重定向,即将中断向量控制寄存器中 SysTick、SVC 和 PendSV 中断服务例程的入口地址映射为 mbedOS 的相应中断处理函数的入口地址.

(2)内存冲突问题主要在 RAM 使用共享栈空间产生.其原因主要有以下两种.其一当在 BIOS 程序中使用 new 或者 malloc 函数进行主动申请内存时,因为 BIOS 的 Heap 段是由下往上使用的,因此若申请空间超出了 BIOS 的 Heap 空间进会涌入 USER 的 data 段和 bss 段,引起程序无法正常执行.另外在 BIOS 中使用的局部变量过大时,如使用一个较大的数组,就会使用过大的栈空间,栈是由上往下使用,容易和 USER 中的堆区进行重叠,造成程序跑飞.

4 GEC 架构下驻留 mbedOS 后启动流程分析

GEC 架构下驻留 mbedOS 后,从芯片上电到用户任务程序调度执行,需经历 BIOS 启动、USER 启动、mbedOS 启动三个阶段.其总体流程如图 7 所示.

4.1 BIOS 的启动流程

BIOS 启动包括从芯片上电至跳转 USER 程序的这部分过程,主要实现以下几个方面的功能:

(1)芯片上电复位后,芯片内部机制首先从 Flash 的 0x00000000 地址中,取出第一个表项的内容,赋给内核寄存器堆栈指针(Stack Pointer, SP),实际上是使用主堆栈指针(Main Stack Pointer,

MSP),完成堆栈指针初始化.

(2)芯片内部机制将第二个表项(即复位处理程序 Reset_Handler 的首地址),赋给内核寄存器程序计数器(Program Count, PC).

(3)芯片内部机制从程序计数器 PC 中取出地址,再从该地址取出机器码,这个机器码就是芯片复位后执行的第一个指令,也就是说,芯片从此开始运行程序了.直观理解就是从启动文件中的 Reset_Handler 函数开始运行了,第一个指令就是关总中断(“cpsid i”),随后运行系统初始化函数.

(4)系统初始化函数进行芯片部分初始化设置、关闭看门狗、系统时钟初始化.

(5)回到 Reset_Handler 中继续执行剩余初始化功能,包括开总中断(“cpsie i”)、将 ROM 中的初始化数据拷贝到 RAM 中、清零未初始化 BSS 数据段,最后调用 BIOS 的 main 函数.

(6)在 main 函数中判断是否有 USER 程序的存在,若有则跳转至 USER 程序,若无则继续在 BIOS 程序中等待 USER 程序的执行.

4.2 USER 的启动流程

USER 程序的启动是从 BIOS 程序跳转开始至 mbedOS 操作系统启动前,大致分为以下几个步骤:

(1)将 ROM 中的初始化数据拷贝到 RAM 中、清零未初始化 bss 数据段,随后执行 USER 的 main 函数.

(2)在 main 函数中通过 SVC 中断获取 API 表的首地址,并进行函数地址重映射.

(3)调用 mbedOS 启动接口函数,进行 mbedOS 的启动.

4.3 mbedOS 的启动流程

mbedOS 的启动主要实现以下几方面的功能:

(1)首先将 SVC、PendSV 和 SysTick 三个中断进行重定向.

(2)调用内核初始化函数实现操作系统信息结构体数据初始化.

(3)设置自启动线程属性,创建自启动主线程,并且将其设置为激活态.

(4)调用内核启动函数,主要是实现空闲线程和定时器线程的创建、设置并且使能时间滴答、将当前堆栈指针 MSP 切换为 PSP 等.

5 mbedOS 驻留后实验测试

测试平台分别采用 NXP 的 S32K144 和 MKL36Z64,相关参数见表 3.并且进行了相应的功

能性、实时性测试.

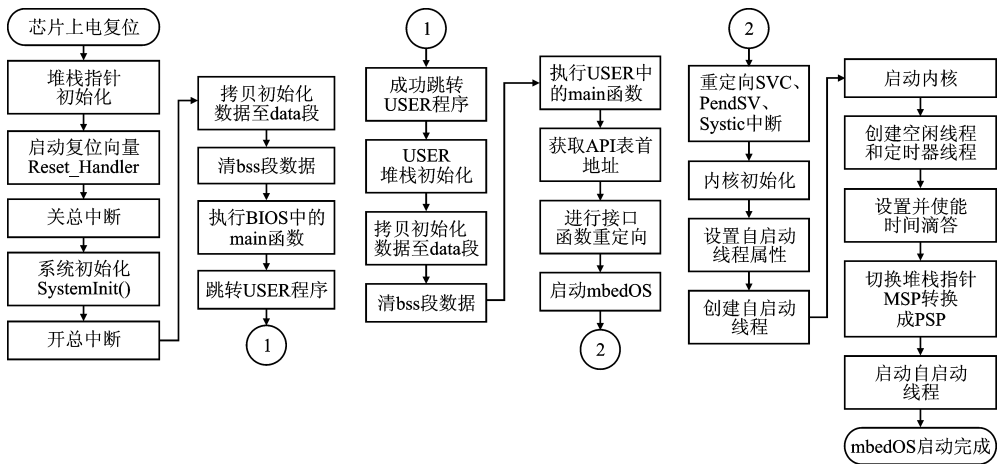


图 6 基于 GEC 框架 mbedOS 启动流程

表 3 硬件测试平台参数

参数	S32K144	MKL36Z64
架构	ARM-V6M	ARM-V7
内核	Cortex-m4	Cortex-m0+
运行最大速率	144MHz	48MHz
Flash 内存	512Kb	64Kb
RAM	64Kb	8Kb

5.1 功能性测试

为验证移植后的系统功能,在 S32K144 和 MKL36Z64 平台上分别设计实现红、绿、蓝三个小灯的轮流闪烁测试,以及通过 UART 向上位机 PC 端发送小灯的闪烁状态.实验表明,程序运行正常.

5.2 系统实时性能测试

mbedOS 操作系统属于实时嵌入式操作系统,因此需要保证系统工程整体实时性不受损坏.本次实时性能测试按照美国 SSC 实验室提出的一个实时性测试标准,主要就嵌入式操作系统实时性能的 3 个关键指标——中断服务响应时间,任务响应时间和任务上下文切换时间进行时性进行测试.其测试结果如表 4 所示.

表 4 mbedOS 实时性能测试

测试项目	测试次数	平均时间/us
中断服务响应时间	200	22
悬挂/恢复任务时间	400	380
上下文切换时间	400	87
任务响应时间	400	249

可以从表 4 中的测试结果看出 mbedOS 在 S32K144 平台下的 3 个关键实时性能指标都在 (us) 级别,可以满足绝大部分的实时要求.

表 5 分别为采用不驻留和驻留方式下 BIOS 和 USER 程序编译时间测试对比.

表 5 驻留与非驻留 USER 程序编译时间

MCU 名称	测试次数	非驻留编译时间/ms	驻留编译时间/ms
S32K144	200	4753	1721
MKL36Z64	200	4069	1523

6 结束语

本文验证了在 GEC 架构下进行 mbedOS 驻留,在驻留实现方法上针对内存划分问题,提出 Flash 划分提出互不干扰原则,RAM 划分采用共享栈或独享栈空间方法来实现空间最大利用.对于 API 接口设计,在静态链接的基础上提出动态链接技术,借助 SVC 实现接口调用.并且分别在 S32K144 和 MKL36Z64 上进行了实验论证,具有可行性,并且缩短了编译时间,加快开发速率,增强了嵌入式应用程序的健壮性与可移植性,具有参考价值.

参考文献:

[1] 王宜怀,张建,刘辉,等.窄带物联网 NB-IoT 应用开发共性技术[M].北京:电子工业出版社,2019:194-195.

[2] WANG H,WANG H,XU W,et al. Development and experiment of a snake-like robot composed of modularized isomorphic joints[C]//2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, 2016: 003160-003165.

[3] JIANG J. Transplantation research of the lwip agreement under the uc/os-ii[C]//2012 5th International Conference on BioMedical Engineering and Informat-

ics. IEEE, 2012: 1376-1379.

[4] WANG H, BIAN H N, ZHANG H X, et al. The research and design of smart home system based on MQX[C]//2015 International Conference on Computer Science and Applications (CSA). IEEE, 2015: 188-190.

[5] NIU S, ZHANG J, WANG B, et al. Analysis and implementation of migrating real-time embedded operating system FreeRTOS kernel based on S3C44B0 processor[C]//2012 Fourth International Symposium on Information Science and Engineering. IEEE, 2012: 430-433.

[6] 孟蓉歌, 张春化, 王佳, 等. 符合 AUTOSAR 标准的 μ C/OS-III 移植[J]. 计算机应用, 2017, 37(2): 248-250.

[7] 马书红. 基于固化操作系统的数学计算机系统设计研究[J]. 电子设计工程, 2018, 26(22): 37-41.

[8] ARM. An introduction to Arm Mbed OS 5[EB/OL]. (2014-03-02)[2019-09-16]. <https://os.mbed.com/docs/mbed-os/v5.14/introduction/index.html>.

[9] ATMADJA W, LIAWATIMENA S, LUKAS J, et al. Hydroponic system design with real time OS based on ARM Cortex-M microcontroller[J]. IOP Conference Series: Earth and Environmental Science, 2017: 012-017.

[10] BALSAMO D, ELBOREINI A, AL-HASHIMI B M, et al. Exploring ARM mbed support for transient computing in energy harvesting IoT systems[C]//2017 7th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI). IEEE, 2017: 115-120.

[11] BOCK C, MARQUARDT M, MARTENS A, et al. Smart Sensors and Actors with BACnet TM and Mbed OS on Cortex-M Microcontrollers [C]//IEEE 2019 5th World Forum on Internet of Things (WF-IoT), 2019: 937-942.

作者简介:

孙亚军 男, (1995-), 硕士研究生. 研究方向为嵌入式系统与物联网. E-mail: 15295085703@qq.com

王 林 男, (1962-), 硕士, 副教授. 研究方向为嵌入式系统与物联网.

王宜怀 男, (1962-), 博士, 教授. 研究方向为嵌入式系统与物联网.

刘长勇 男, (1974-), 本科, 副教授. 研究方向为嵌入式计算.

程宏玉 男, (1993-), 硕士研究生. 研究方向为嵌入式系统与物联网.

(上接第 11 页)

[5] ZHU Y, HU Y, YANG T, et al. Reliability-Optimal Offloading in Multi-Server Edge Computing Networks with Transmissions Carried by Finite Blocklength Codes[C]//2019 IEEE International Conference on Communications Workshops (ICC Workshops). Shanghai, China; IEEE, 2019: 1-6.

[6] ROCKAFELLAR R T, URYASEV S. Conditional value-at-risk for general loss distributions[J]. Journal of banking & finance, 2002, 26(7): 1443-1471.

[7] ARTZNER P, DELBAEN F, EBER J M, et al. Coherent measures of risk[J]. Mathematical finance, 1999, 9(3): 203-228.

[8] SARYKALIN S, SERRAINO G, URYASEV S. Value-at-risk vs. conditional value-at-risk in risk management and optimization[M]//State-of-the-art decision making tools in the information-intensive age. Inform, 2008: 270-294.

[9] ROCKAFELLAR R T, URYASEV S. Optimization of conditional value-at-risk[J]. Journal of risk, 2000, 2: 21-42.

作者简介:

赵若海 男, (1995-), 硕士研究生. 研究方向为边缘计算、超可靠低延迟通信.

杨涛(通讯作者) 男, (1970-) 博士, 副教授. 研究方向为智能信号处理和超可靠低延迟通信在大规模无线接入网络中的应用.
E-mail: taoyang@fudan.edu.cn

胡 波 男, (1968-) 博士, 教授. 研究方向为数字信号处理、数字通信和系统设计.