基于 AUTOSAR4.2 标准嵌入式多核操作系统的实现

朱 元[1]，李 超[1]，吴志红[1]，雷小军[2]
（1.同济大学中德学院；2.上海大郡动力控制技术有限公司，上海 200000）

摘要：介绍了 AUTOSAR 标准产生的背景及软件架构，阐述了多核处理器的启动、AUTOSAR 多核操作系统的启动与实现过程。以 Vector 的软件配置工具配置符合 AUTOSAR 标准的嵌入式多核操作系统，在英飞凌 AURIX/TC275 三核单片机上运行该操作系统从而验证 AUTOSAR 多核操作系统的可行性。

关键词：AUTOSAR 标准；多核处理器；多核操作系统
中图分类号： **文献标识码**：A **文章编号**：1673-1131（2017）05-0055-05

An Implementation of Embedded Multi-Core Operating System Base on AUTOSAR 4.2 Standard

Abstract: This paper introduces the context, in which the AUTOSAR Standard developed, discusses the startup of Multi-Core Microprocessor and its OS, describe the steps to implement AUTOSAR Multi-Core OS. With the software tools supplied by Vector a Multi-Core OS base on AUTOSAR Standard 4.2 was implemented, which run in the TriCore processor Infineon AURIX/TC275, so the AUTOSAR Multi-Core OS is realizable.

Key words: AUTOSAR Standard；Multi-Core processor；Multi-Core OS

0 引言

近 30 年来，嵌入式微控制器的种类愈加繁多，有的汽车上应用了多达 70 多个微控制器。汽车上种类繁多的微控制器使得汽车软件的可移植性变差，开发人员被迫将大量的工作花费在开发和调试软件上。为解决该问题，AUTOSAR（AUTomotive Open Systems Architecture，汽车开放系统架构）组织建立一个开放的、标准化的软件架构，使汽车嵌入式软件的接口和开发过程标准化，旨在提高汽车嵌入式软件的可移植性，方便汽车软件的更新和升级，使软件开发朝着模块化的方向发展。

考虑到汽车上电磁波干扰因素，汽车处理器的频率不能过高；另外，由于成本、功耗、空间的限制，汽车上的微控制器数量不应过多。多核处理器能够在不提高工作频率、降低成本和功耗、节约空间的前提下提高运算效率，其工作效率不仅与处理器核的数量有关，也与操作系统（OS：Operating System）和任务调度算法相关，所以多核处理器逐渐成为车用芯片的主流，多核操作系统和任务调度算法也成为了研究的热点。

AUTOSAR 从 4.0 版本开始支持多核操作系统，英飞凌的 AURIX（Automotive Realtime Integrated Next Generation Architecture）单片机是性能较高的多核处理器，应用范围比较广，因此，基于 AUTOSAR4.2 标准嵌入式多核操作系统的实现具有一定实用价值。本文以 Vector 的配置工具和英飞凌 AURIX/TC275 评估板为实验工具，阐述符合 AUTOSAR 标准的多核操作系统的实现。

1 AUTOSAR 软件架构下的的多核操作系统的相关概念

1.1 AUTOSAR 软件架构概述

如图 1 所示，AUTOSAR 软件架构大致可分为三层：

最上层为应用层（应用层 Application Layer），应用层包含许多 AUTOSAR 软件组件（SWC：Software Component）。一个 AUTOSAR 软件组建通过定义好的端口与其它 AUTOSAR 软件组件进行交互。软件组件的行为通过一个或多个可运行实体（Runnable）实现。

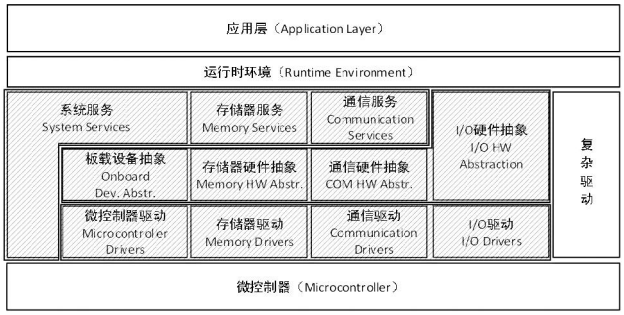

图 1 AUTOSAR 软件架构示意图

中间为运行环境层（RTE：Runtime Environment），它不依赖 ECU，是最小可运行实体（Runnable）的实时运行环境，实现了特定 ECU 上的虚拟功能总线，支持软件组件之间、基础软件间以及软件组件与基础软件之间的通信，其中包括基础软件的通信和通信服务。

RTE 之下为基础软件层（BSW：Basic Software），主要由四部分组成，服务层（Service Layer）、即 ECU 抽象层（ECU Abstraction Layer）、微控制器抽象层（Microcontroller Abstraction Layer）、以及复杂驱动（Complex Drivers）。服务层的系统服务包括 OS、BswM（基础软件模式管理器，Basic software mode Manager）、EcuM（Ecu 状态管理器，Ecu state Manager）等。

通过 AUTOSAR 的分层结构和接口的表转化，使主机厂和供应商之间的分工更加明确，达到 ECU 软件开发与具体硬件相脱离的目的，上层应用策略开发的人员可以更加专注于软件功能的开发，而不用纠缠于底层的细节。

1.2 AUTOSAR OS 概述

AUTOSAR OS 是在 OSEK OS 的基础上进行修改和扩展的，两者既有共性也有区别。

1.2.1 任务的符合类

AUTOSAR OS 定义了 4 种任务的符合类，每个符合类都包含了一个由应用指定的符合类，各符合类及其属性见表 1：

表 1 任务的符合类

属性	BCC1	BCC2	ECC1	ECC2
基本任务激活数	1	≥1	1	≥1
每个优先级的任务数	1	≥1	1	≥1
基本任务	Yes	Yes	Yes	Yes
扩展任务	No	No	Yes	Yes

表 1 中 BCC1 为基本符合类 1，BCC2 为基本符合类 2，ECC1 为扩展符合类 1，ECC2 为扩展符合类 2。基本任务激活数是指在一个任务没有被执行完毕时是否可多次激活，为 1 表示多次激活无效；≥1 表示多次激活有效，同等优先级的任务被放入队列等待执行。

1.2.2 基本任务与扩展任务

基本任务有三种不同的状态，其状态之间的切换见图 2：

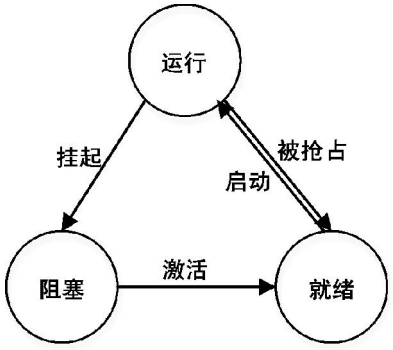

图 2 基本任务各状态之间的切换

阻塞状态下的任务可由 API（Application Programming Interface，应用程序编程接口）函数 ActivateTask()或 ChainTask()激活从而进入到就绪状态，调度器将处于就绪状态下优先级最高的任务启动从而进入到运行状态，运行状态下的任务由 API 函数 TerminateTask()挂起从而进入到阻塞状态或者被更高优先级的任务抢占进入到就绪状态。

扩展任务有四种不同的状态，其状态之间的切换见图 3：

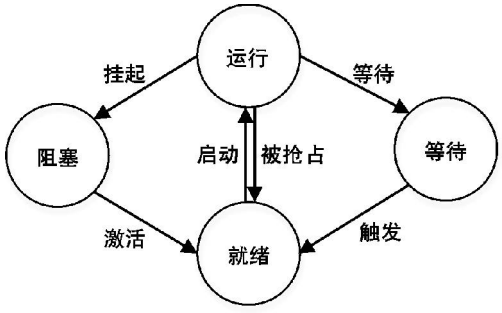

图 3 扩展任务各状态之间的切换

和基本任务相比，扩展任务多了一个等待状态，当运行状态下的任务需要等待一个事件时，任务进入等待状态，事件发生后进入就绪状态。

任务的优先级是静态分配的，但是在某些特定情况下，任务抢占资源会发生死锁，或者任务在运行不能被其他任务抢占。前者的解决方法是优先级天花板模式，后者是通过资源调度的方法实现的，两者的本质是任务获得资源时任务的优先级提升到资源的优先级。

2 多核操作系统的启动

2.1 嵌入式多核处理器的启动

嵌入式系统在上电、复位或唤醒后，首先进入 Boot Mode，进行锁步核的开或者关，对 Falsh 和 RAM 进行初始化，然后进入用户程序，用户程序是从 cstart 函数开始执行的，在多核嵌入式系统中通常是先进行主核的 cstart，主核自身进行部分初始化后将从核从 HALT 状态激活；然后主核和从核在完成各自必要设置后分别进入各自的 main 函数。图 4 是英飞凌 AURIX/TC275 三核处理器的启动流程：

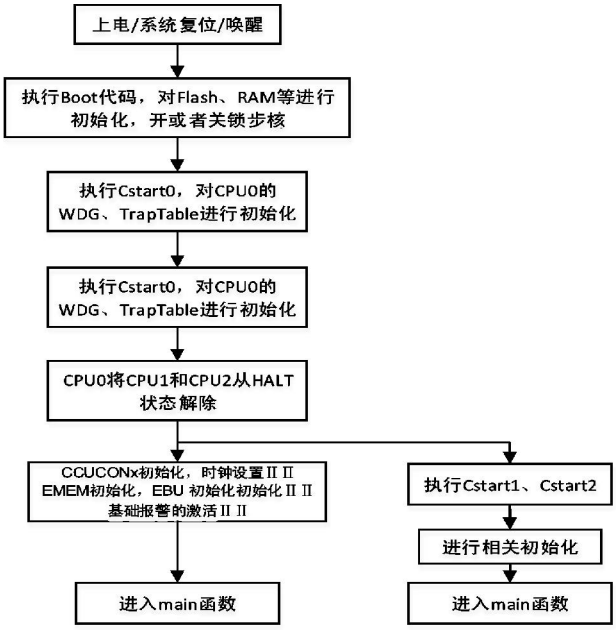

图 4 英飞凌 AURIX/TC275 三核处理器的启动流程

2.2 符合 AUTOSAR4.2 规范的多核操作系统的启动

进入主核的 main()函数之后，进行操作系统的启动。操作系统启动可分为三个阶段，既 PreStartOS，StartOS，PostStartOS，在 ATUOSAR 规范中这三个阶段是通过调用 EcuM_Init() 函数来实现的。

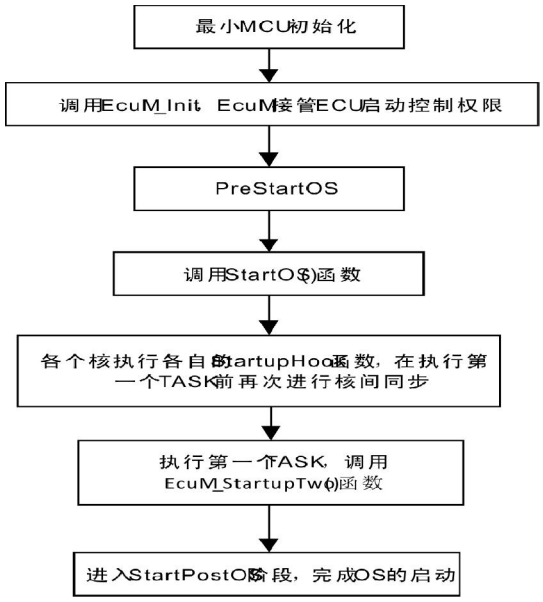

图 5 多核操作系统启动流程

在 PreStartOS 阶段系统主要进行如下工作：设置所有可编程的中断的优先级，执行 EcuM_AL_DriverInitZero()，该函数对 Mcu 进行不完全的初始化从而为硬件提供时钟信号，调

用 EcuM_DeterminePbConfiguration ()，从而决定哪些 post-build 设置需要被使用，检验数据一致性，调用 Callout EcuM_AL_DriverInitOne()函数，初始化在 OS 启动后需要用到的芯片驱动，获取复位原因，设置 ShutdownTarget2。

在 StartOS 阶段，选择启动模式，操作系统进行核间同步，各个核调用各自的 StartupHook，再次进行核间同步，执行操作系统的第一个任务，该任务被称为初始化任务，它调用 EcuM_StartupTwo()函数，系统进入 PostStartOS 阶段。

PostStartOS 完成两个部分的初始化：一是初始化 SchM（BSW Scheduler，基础软件调度器），二是调用 BswM_Init () 函数从而对 BswM 初始化。操作系统初始化流程如图 5 所示。

3 通过实验实现 AUTOSAR 多核操作系统

3.1 实现工具

AUTSOAR OS 可以通过商业配置工具来实现，本文主要运用 Vector 提供的 DaVinci Developer 和 DaVinci Configurator 进行多核 OS 的配置，用 EB Tresos 配置的 IO 端口和周期定时器模块 Gpt，加入了 LED 灯以方便观察操作系统运行状况，系统运行于英飞凌 AURIX/TC275 上。开发流程如图 6 所示：

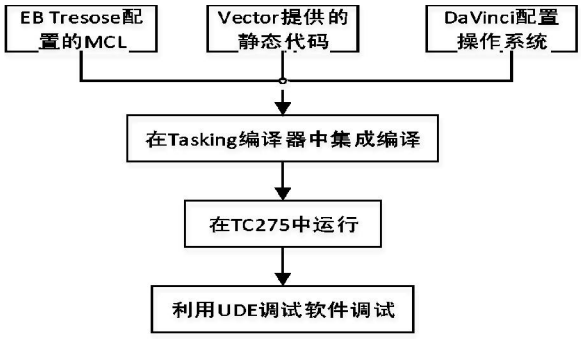

图 6 AUTOSAR OS 开发流程

Vector、Infineon（英飞凌）、EB 均为 AUTOSAR 组织的主要成员。

3.2 操作系统启动的实现

如 2.2 节所述，操作系统启动前应进行一系列的初始换，本文中操作系统启动前的初始化函数如表 2 所示：

表 2 操作系统启动前的初始化函数

初始化函数	初始化阶段
BswM_InitMemory()	PreStartOS，EcuM_AL_DriverInitZero
Rte_InitMemory()	PreStartOS，EcuM_AL_DriverInitZero
Dio_Init()	PreStartOS，EcuM_AL_DriverInitOne
Gpt_Init	PreStartOS，EcuM_AL_DriverInitOne
Icu_Init	PreStartOS，EcuM_AL_DriverInitOne
Mcu_Init	PreStartOS，EcuM_AL_DriverInitOne
Port_Init	PreStartOS，EcuM_AL_DriverInitOne
IrqGtm_Init	PreStartOS，EcuM_AL_DriverInitOne

3.3 操作系统中 TASK 和 Runnable 的映射

实验总体要求如下，因 TC275 有三个核，故需验证操作系统是否在三核中均可运行。Core0 为主核，操作系统从该核开始启动，ECU 的初始化也由该核完成，为便于观察该核是否正常运行，可使该核定周期的翻转 IO，以实现 Led 的闪烁，从而便于观察。Core1 中分配定周期触发的 TASK，Core2 中仅分配 IdleTask3。系统总体框图如图 7 所示：

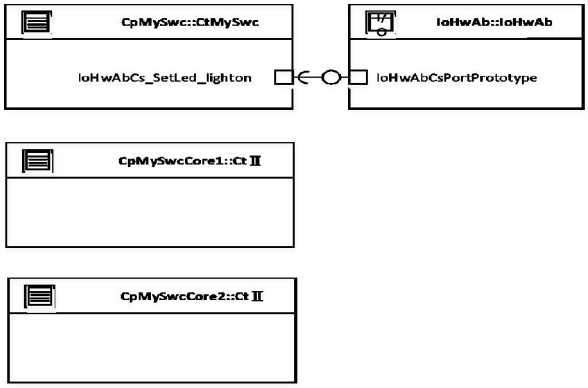

图 7 多核操作系统总体框图

除了用户自己定义的 Runnable 外操作系统还应执行一些特殊的 Runnable，如获取 ShutdownTarget 及 Shutdown 的原因，定周期执行 EcuM_MainFuction()函数以检查唤醒请求和更新报警时钟等，这些 Runnable 也需要运行在主核中，为此定义了一个专门用于调度的 TASK，命名为 ScheduleTask。TASK 和 Runnable 之间的映射关系如表 3 所示：

表 3 TASK 与 Runnable 的映射关系

TASK	Runnable	SWC
ScheduleTask	EcuM_MainFuction GetShutdownTarget ……	CtMSwc
Core0_InintTask	无	CtMySwc
Core0_Task_SchM	LightControl	CtMySwc
Core0_Task_1ms	Runnable_Core0_1ms	CtMySwc
Core0_Task_2ms	Runnable_Core0_2ms	CtMySwc
Core0_Task_5ms	Runnable_Core0_5ms	CtMySwc
Core0_IdleTask	无	CtMySwcCore1
Core1_Task_100ms	Core1_Runnable_100ms	CtMySwcCore1
Core1_IdleTask	无	CtMySwcCore2
Core2_IdleTask	无	CtMySwcCore2

虽然没有 Runnable 映射到 Core0_InitTask，但该 Task 执行一条非常重要的函数，即 EcuM_StartupTwo ()，如下段代码所示，从而进入 PostTaskOS，该 TASK 是操作系统执行的第一个 TASK。

```
TASK(Core0_InitTask)
{
    EcuM_StartupTwo();
    Rte_Start();
    TerminateTask();
}
```

3.4 部分代码展示及注释

上节以展示了 TASK 的运行代码，其余 TASK 在其结束后也要调用 TerminateTask ();使自己结束运行，下面分别展示运行实体的代码、IO 抽象层的代码和中断服务函数的代码。运行实体代码：

```
FUNC(void, CtMySwc_CODE) LightContro (void)
{
    static boolean lighton = 0; //定义静态变量
    lighton=!lighton;
    if(E_OK == Rte_Call_IoHwAbCS_SetLed_lighton_IoHwAbOperation(&lighton))
    {}//将取反后的值写入地址
}
```

IO 抽象层代码：

```
FUNC(Std_ReturnType, IOHWAB_CODE) IoHwAb_IoHwAbCSPortPrototype_IoHwAbOperation(
    P2VAR(uint16, AUTOMATIC, RTE_IOHWAB_APPL_VAR) IoHwAb_lighton
)
{
    Dio_WriteChannel(DioConf_DioChannel_DioChannel_0, *IoHwAb_lighton);
    //将指针变量值写入 Port 口 P33.8
    return E_OK;
}
```

周期性中断代码：

```
void GptChannel0_Notification()
{
    Dio_WriteChannel(DioConf_DioChannel_DioChannel_1,  !Dio_ReadChannel(DioConf_Dio
Channel_DioChannel_1));
//周期性读取端口 P33.9 电平取反后写入自身端口 P33.9
}
```

3.5 实验结果分析

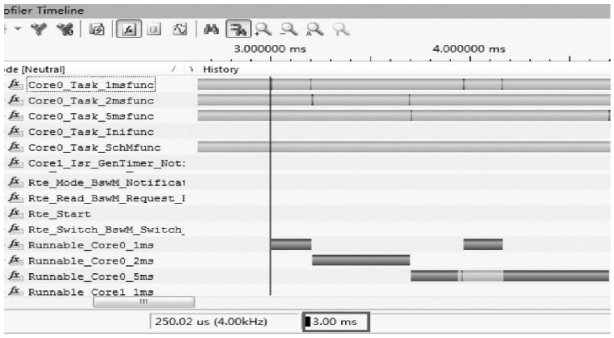

图 8 TASK 与 Runnable 的对应关系

图中的短线代表任务、Runnable 或者中断发生了切换，从图 8 可以看出，在 3.00ms 时 Core0_Task_1msfunc 会被触发，TASK 被触发后调用了函数 Runnable_Core0_1ms，当该函数执行完成以后，Core_Task_2ms 被触发，然后该 Task 调用函数 Runnable_Core0_2ms。以此类推，其余 TASK 与 Runnable 的对应关系类似。

由此可得出如下结论：

Runnable 被成功地映射到了相应的 TASK。

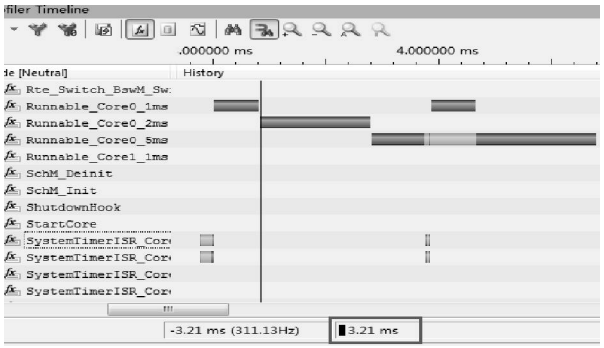

图 9 Runnable 和系统中断的工作状态

从图 9 中可以看出，Runnable_Core0_1ms 从 3.01ms 开始执行，至 3.21ms 结束，执行时为 200μs，符合任务需求表要求；同理可以看出，Runnable_Core0_2ms 执行时为 500μs，也满足项目要求；Runnable_Core0_5ms 从 3.71ms 开始执行，至 3.95ms 被系统中断 SystemTimerISR_Core0 打断，在 3.97ms 又被比它优先级高的 TASK 所打断 (因为这里的 Runnable 是被 TASK 所调用，所以 Runnable 的调用就代表了任务的调度)，此时程序执行了 3.97-3.01=0.96ms 约为 1ms，近似为 Core0_Task_1ms 的执行周期，至 4.16msRunnable_Core0_5ms 重新得到执行。

由此可得出如下结论 :中断可以打断 TASK ;优先级高的任务可以打断优先级低的任务；当没有更高优先级任务或者中断时，被打断的任务继续得到执行。

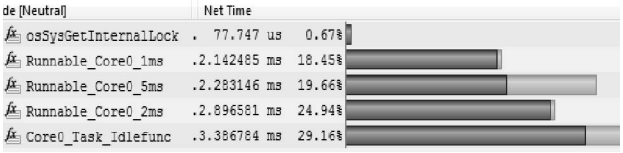

图 10 部分运行实体运行所占的时间百分比

从图 10 可以看出，中 Runnable_ Core0_1ms 执行时间为 18.45%，Runnable_Core0_2ms 执行时间为 24.94%, Runnable_Core0_5ms 执行时间为 19.66%，idle Task 的执行时间为 29.16，总和为 92.21%，其余时间为 AUTOSAR OS 本身占用的时间，如 :检查操作系统运行模式、检查使能或关闭中断请求、清除事件等 API 函数需占用一定时间。

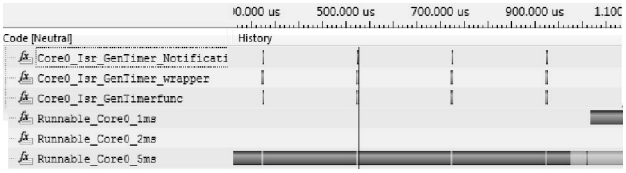

图 11 定时中断

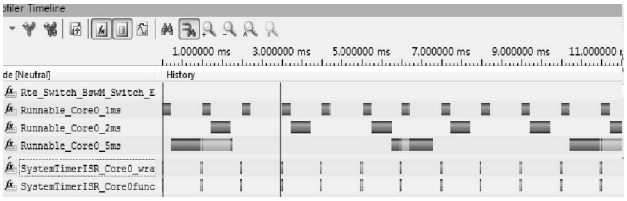

图 12 系统中断

从图 11、图 12 可知定时中断的时间的周期为 200μs，系统中断周期为 1ms，满足需求文档中提出的要求。当中断发生时任务被打断，这是因为在 AUTOSAR OS 中中断可以打断任意优先级的任务。

4 结语

本文对 AUTOSAR 多核操作系统的启动和实现进行了较为细致的探讨，为满足功能安全的需求，还需研究 AUTOSAR 多核操作系统的调度表、时间保护机制、内存保护机制 ;为充分发挥多核处理器的并行计算能力，还需要对操作系统性能指标、任务分配和调度算法进行深入的研究。

参考文献：

[1] AUTOSAR Administration. Specification of Operating System AUTOSAR Release 4.2.2 [S/OL] http://www.autosar.org/standards/classic-platform/release-42/software-architecture/system-services/

[2] AUTOSAR Administration. Specification of ECU State Manager AUTOSAR Release 4.2.2[S/Ol] http://www.autosar.org/standards/classic-platform/release-42/software-architecture/system-services/

[3] AUTOSAR Administration. Specification of Basic Software Mode Manager AUTOSAR Release 4.2.2[S/Ol] http://www.autosar.org/standards/classic-platform/release-42/software-architecture/system-services/

基于 Android 的汽车 TPMS 实现

杨文涛[1]，金 格[1]，廖堉哲[1]，王灵广[1]，冯则坤[1]，刘长华[2]，戴建飞[2]
（1.华中科技大学光电学院，湖北 武汉 430074；2.南昌工控电装有限公司，江西 南昌 330001）

摘要：随着汽车行业的快速发展，以及车联网的逐渐渗透，人们在享受丰富的车载用户体验的同时，对汽车安全性能的要求越来越高，而胎压是汽车安全隐患因素之一，所以实时监测胎压必不可少。其次，由于 WinCE 系统的 CPU 效率以及对网络支持能力的瓶颈，越来越不能满足人们的需求，而当今主流的 Android 操作系统很好地解决了这些难题，所以基于 Android 车载设备必将是未来车载系统的发展趋势。该系统是基于有源式的汽车胎压监测系统(TPMS)，通过相关传感器设备能实时采集汽车轮胎胎压、温度和加速度等运行状态的数据，并通过 CC1101 射频模块将数据传送至胎压监视中心，由胎压监视中心显示这些参数值，并能够对异常情况发出警报提示信息。

关键词：Android；TPMS；CC1101 射频模块
中图分类号：U469.3 **文献标识码**：A **文章编号**：1673-1131（2017）05-0059-03

0 引言

随着汽车的逐渐普及，汽车的安全性问题是汽车技术发展过程中最重要的问题之一，高速行驶的车辆，其最难预防的故障便是轮胎故障。据统计，全世界发生的汽车交通事故中有 1/3 是轮胎故障导致的，所以轮胎压力监测系统(TPMS)也便愈来愈受到重视。TPMS 系统作为安全预警系统，对于轮胎故障的监控和报警功能大大降低轮胎出现故障的几率，对于防止爆胎有着重要的作用。而爆胎又是高速公路事故的首要原因，因此 MTPMS 系统的重要性已经越来越被大众认可和接受[1]。

传统的车载监控设备大多数采用 WINCE 系统，但由于 WinCE 系统的运行效率较低、应用程序扩展性较差以及对网络支持能力很弱，越来越不能满足人们的需求。当今主流的 Android 操作系统很好地解决了这些难题，所以基于 Android 车载设备必将是未来车载系统的发展趋势。由此本文提出了一个基于 Android 系统的汽车胎压实时监测系统的实现，系统可以实时监测汽车轮胎压力，温度、加速度等数据。

1 系统原理与设计

1.1 系统原理

TPMS 的工作原理是在每个轮胎的胎骸上安装一个胎压数据采集模块组件，胎压数据经过 MCU 处理后再通过 RF 电路无线信号发射出去，车载胎压监测中心的无线接收模块接收到无线信号经过处理还原出胎压数据，再传送给 ARM 上层应用程序，最终实现胎压数据显示和报警，达到胎压监测的目的，其工作原理图见图 1。

一般情况下一辆汽车有四个轮胎，总共需要安装四个胎压数据采集端，因此它们与胎压监测中心形成多对一的数据通讯关系，胎压监测中心能实时监视各个轮胎工作状况[2]。

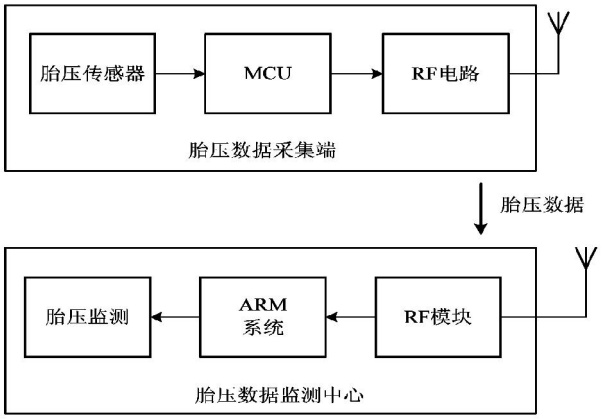

图 1 系统工作原理图

1.2 系统设计

根据上述原理可知，系统设计包含胎压数据采集端和胎压监测中心两个部分，其中胎压数据采集端能够获取汽车在行驶或静止状态下，轮胎的胎压、温度、加速度等数据。其硬件结构包含胎压、温度、车轴加速度等多个传感器单元以及无线信号发送单元。为了使这些设备单元协调工作，需要通过单片机 MCU 有效地控制，完成各类传感器数据的采集、封装和发送。系统灵活地支持各种工作模式以降低 MCU 的功耗。为了降低开发难度，本设计方案直接选用基于 FXXX 芯片胎压传感器解决方案。

胎压监测中心功能要求：一方面能够接收多个胎压数据采集端发送的多路无线信号并且从中解调出胎压数据，实现胎压数据处理；另一方面，需要提供交互良好的操作界面以及其它丰富的扩展功能，如网络服务、多媒体影音等。实现这些功能需要胎压监测中心有强大的软、硬件系统作支撑，而传统的单片机不能满足这些要求。因此，本设备硬件平台采用基于 ARM 的 Exynoss 4412 处理器核心板，操作系统采用 Android 系统，并在此基础上开发 App 实现胎压监测应用，系统设计架

[4] 程露，朱元. 基于 Aurix~(TM)的 AUTOSAR 多核应用实现[J]. 自动化技术与应用，2016(7):27-31.
[5] 张翟辉，朱元，王民. 基于 Aurix 的 AUTOSAR 多核操作系统的实现[J]. 工业控制计算机，2016(3):43-45.
[6] 钟再敏，张允. 基于 AUTOSAR 4.0 标准多核软件系统实现研究[J]. 机电一体化，2016(1):12-17+22.

基金项目：国家重点重点研发计划课题电机控制器功能安全（2016YFB0100804）资助。

作者简介：朱元，同济大学中德学院副教授；李超，同济大学中德学院 14 级硕士；吴志红，同济大学中德学院教授；雷小军，上海大郡动力控制技术有限公司总工程师。