第22卷 第1期
2007年3月

青岛大学学报(工程技术版)
JOURNAL OF QINGDAO UNIVERSITY (E&T)

Vol.22 No.1
Mar. 2007

文章编号:1006-9798(2007)01-0020-06

嵌入式系统开发平台硬件抽象层的研究与设计*

张 乐, 邵峰晶, 孙仁诚
(青岛大学信息工程学院, 山东 青岛 266071)

摘要: 针对嵌入式系统开发平台构建中的硬件环境差异问题,给出了一种高移植性的硬件抽象层设计方案,并介绍了实现方法。该硬件抽象层给出底层封装规范并向上层软件设计提供操作接口规范,屏蔽硬件环境差异,增强了嵌入式系统开发平台的开放性,可有效地缩短嵌入式产品的开发周期。

关键词: 嵌入式系统开发平台; 硬件抽象层; 可移植性; 软硬件协同设计
中图分类号: TP311 **文献标识码**: A

嵌入式系统开发平台是缩短嵌入式产品开发周期、提高系统资源复用率的一种有效解决方案。但因其专用性强、体积小以及外围设备种类繁多等特性,决定了其应用的硬件环境的较大差异性。因此,如何科学合理地设计开发平台并使其能够适用于多种不同的应用环境是嵌入式系统发展中的关键问题。

将底层与硬件相关的部分单独设计为硬件抽象层(HAL, hardware abstraction layer)是保证操作系统的稳定性和兼容性的关键技术。硬件抽象层是:"将硬件的差别与操作系统其他部分隔离开来的软件层次,通过采用硬件抽象技术使得操作系统可以对同种类的不同系列的硬件采用统一的操作手段进行控制"[1]。可见,提出硬件抽象层的目的是通过建立向上的接口规范,实现硬件相关和硬件无关的两部分程序代码的隔离,对硬件无关部分屏蔽具体硬件的差异,即通过硬件抽象层可为上层软件提供一个没有硬件特性的接口。关于硬件抽象层目前有许多较为成熟的设计方案,如 WindowsNT HAL[2]和 RTHAL(Real Time Hardware Abstracttion Layer)[3]。WindowsNT HAL 以动态链接库的形式提供了一组面向平台的接口,从而将 WindowsNT 操作系统与其所依赖的基本硬件进行了分离,并负责处理底层的输入输出、中断、硬件缓存以及多处理机间的相互通讯[4]。RTHAL 作为 RTAI 的硬件抽象层为 Linux 提供实时系统调度方案和中断处理所需进行的硬件访问接口,同时兼容多种体系结构的硬件平台,从而使得 RTAI 得到了广泛的应用。但 WindowsNT HAL 是针对 PC 机进行的设计,没有为嵌入式系统做特别优化;而 RTHAL 的设计结构、提供的功能以及所定义的服务接口与 Linux 密切相关,使得该硬件抽象层相对封闭而无法被其它嵌入式操作系统利用。其它硬件抽象层设计方案也大都存在类似的问题。本文通过给出底层封装规范和上层接口规范,设计并实现了具有较强通用性的嵌入式系统开发平台硬件抽象层(ESP-HAL, embedded system platform HAL),可支持不同应用环境中嵌入式系统开发平台的快速构建。

1 ESP-HAL 设计概要

ESP-HAL 是对硬件的一种抽象,即从物理硬件到逻辑硬件的一种转换,并以一组统一的硬件相关功能服务接口的方式为上层系统内核提供一套逻辑意义上的硬件环境,而该层本身实现了从逻辑功能到物理操作的转换。在嵌入式系统开发平台中,ESP-HAL 是直接面向底层硬件设备的系统软件层,其所处的位置如图 1 所示。

嵌入式系统开发平台在完成系统软硬件设计需求分析之后,只需进行 ESP - HAL 接口的定义,即可使

* 收稿日期: 2007-03-12
基金项目: 国家高技术研究发展计划(863)计划)基金(2003AA1Z1110)
作者简介: 张乐(1983-),江西南昌人,硕士研究生,主要研究方向为嵌入式系统设计。E-mail: javale@163.com

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

硬件和软件的设计和测试工作在统一的接口上并行进行,有效地分离硬件模块和软件模块设计中的错误。该种开发模式(图 2 所示)支持嵌入式系统的软硬件协同设计,可缩短系统的开发周期并提高系统的可靠性。

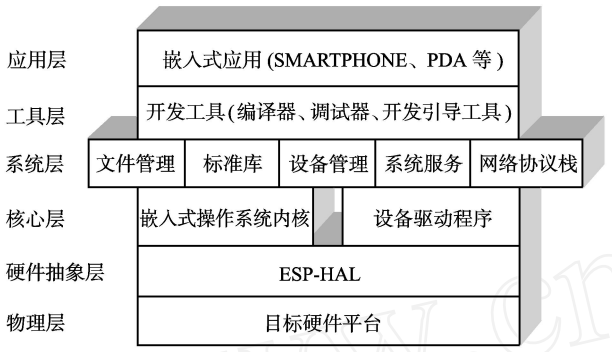

图 1　嵌入式系统开发平台层次模型

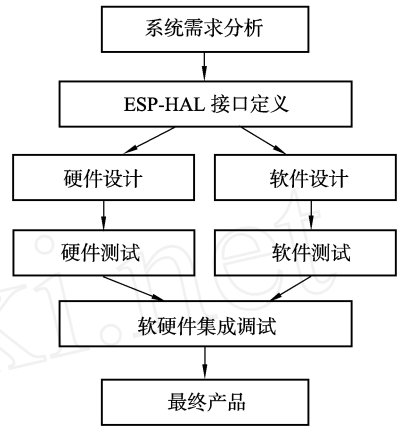

图 2　基于 ESP-HAL 的系统开发模式

2　ESP－HAL 层次结构

为实现硬件相关和硬件无关两部分程序代码的隔离,对硬件无关部分屏蔽硬件差异,使得硬件抽象层可为上层软件提供一个没有硬件特性的接口,将 ESP-HAL 进一步划分为 5 个子层:通用功能层、体系结构层、处理器变体层、外围抽象层以及辅助功能层(图 3 所示),各子层的详细描述如表 1 所示。

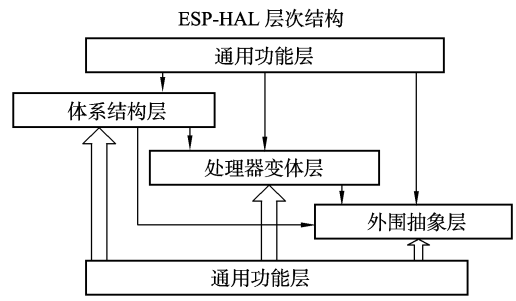

图 3　ESP-HAL 层次结构

除为系统内核和应用程序提供统一的功能服务接口外,为了便于扩展和移植到其他硬件平台,在 ESP-HAL 各子层之间还定义了统一的接口调用。这些接口采用 C/C++ 语言进行编码,并定义为宏的方式。由于宏定义在预编译时进行替换,没有执行时的性能损失(如采用函数调用则需要返回地址、参数压栈等过程),既可确保实现接口对底层硬件端口寄存器的访问,又可提高系统的执行效率。

表 1　ESP－HAL 子层详细描述

层次类型	描　　述	功能概要
通用功能层	ESP-HAL 中所有模块共享的配置和函数	基本数据类型、寄存器访问操作、IO 设备读写操作、原子操作等基本功能的实现
体系结构层	实现该种体系结构的特殊功能、为系列结构层和平台抽象层提供某些功能的默认实现	体系结构的 debug 功能、中断向量的定义和处理程序、Cache 定义和控制、上下文切换程序、系统初始化汇编程序、配置选项等功能的实现
处理器变体层	同种体系结构包含的不同系列处理器(如 ARM 系列具有 ARM7、ARM9 等不同版本的 CPU)的功能描述。	不同系列 cpu 对体系结构进行扩展的程序(Cache、中断)、配置选项、片内设备的驱动程序、测试程序等功能的实现
外围抽象层	包含平台专有的功能和配置选项	具体硬件平台初始化程序、平台内存布局、配置选项(处理器时钟、编译选项)、外围设备访问控制、平台用于扩展体系结构和 cpu 功能的程序等功能的实现
辅助功能层	实现提供辅助功能的模块的描述	常用 debug 功能、测试程序、ROM 监视调试等功能的实现

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

3 ESP-HAL 功能结构

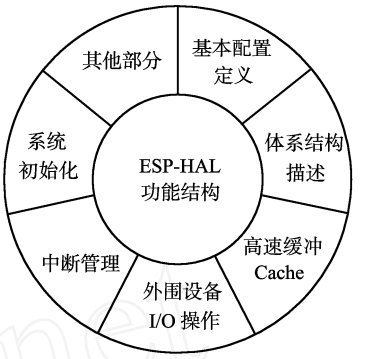

图4 ESP-HAL 功能结构示意图

ESP-HAL 的功能服务接口的详细设计主要分为:基本配置定义、体系结构描述、高速缓存 Cache、外围设备 I/O 操作、中断管理、系统初始化等,其功能结构示意图如图 4 所示。

3.1 基本配置定义

基本配置定义包括在对 ESP-HAL 进行编译时需要使用的与基本体系结构相关的一些特性定义,如数据类型、字节排列顺序等。

1) 字节排列方式定义

对于不同类型的处理器,目标系统的字节排列方式是不同的。目前常见的字节排列方式分为小端格式和大端格式。在 ESP-HAL 中采用宏 BYTEORDER 即

enum BYTEORDER (LittleEnd, BigEnd)

对目标系统的字节排列方式进行定义,取值可以为 LittleEnd(大端格式)或 BigEnd(大端格式)。

2) 基本数据类型

对于不同的处理器,C 语言所对应的数据类型不尽相同。为了保证服务接口的一致性,在 HAL 中定义一套数据类型,如表 2 所示。

表2 ESP-HAL 中基本数据类型定义

宏 名	描 述	宏 名	描 述
BOOLEAN	布尔型	CHARACTER	字符类型
INT8(U/S)	8位(无/有)符号整型	STRN	字符串类型
INT16(U/S)	16位(无/有)符号整型	ARRAY	数组类型
INT32(U/S)	32位(无/有)符号整型	ATOMIC	原子类型
FLOAT	单精度浮点型	HWORD	半字类型
DOUBLE	双精度浮点型	WORD	字类型

3.2 体系结构描述

体系结构描述是在 ESP-HAL 中对处理器的基本结构进行定义,包括现场环境存储格式、线程现场环境初始化、现场切换、位索引操作、堆栈大小和地址转换等。

1) 寄存器存储格式

在 ESP-HAL 中采用 Reg_Saved 结构对保存在堆栈内的现场环境(主要是当前的 CPU 状态)进行描述。在线程现场切换、中断处理过程中将保存当前的现场环境。对于不同体系结构的处理器,需要保存状态的数量并不相同。该结构的具体实现需针对特定的硬件平台进行,如,针对 ARM 体系结构的寄存器存储格式为:

```
typedef struct Reg_saved
{ ……/ *以上的具体实现省略 */
WORD   vector;/ *中断向量表中元素的个数 */      WORD   mdr; / *分频寄存器 */
WORD   mcvf;/ * MAC 溢出标志 */                 WORD   mcrl; / * MAC 寄存器低位地址 */
WORD   mcrh; / * MAC 寄存器高位地址 */          WORD   sp; / *堆栈指针 */
WORD    psw; / *程序状态字 */                   WORD   pc; / *程序计数器 */
} Reg_saved;
```

2) 线程现场环境初始化

在 ESP-HAL 中,对线程现场环境进行初始化是由 CONTEXT-INIT 宏完成的。该项初始化工作是

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

实现线程现场切换的基础,其定义为:

CONTEXT_INIT(sp ,data ,addr) ;

其中,sp ,data 和 addr 分别是线程堆栈当前指针所在的位置、传递的参数和入口函数的地址。

3)　线程现场切换

线程现场切换由宏 CONTEXT-SWITCH 进行控制,其定义为:

CONTEXT_SWITCH(src ,dec) ;

是将当前线程的 CPU 状态保存到它的堆栈,并从 dec 指定的位置加载新线程的状态。

3.3　Cache 控制描述

在 ESP-HAL 层提供对 Cache 控制的支持。对于不同的硬件平台可能在不同的层次实现对 Cache 的控制,通常体系结构层对基本的 Cache 控制宏进行定义,而对于某些特殊的硬件环境可以在处理器变体层对 Cache 进行重新覆盖定义。在 ESP-HAL 层提供了几种对 Cache 进行管理的服务接口。

1)　全局 Cache 控制

全局 Cache 控制包括 Cache 使能、关闭、失效以及加/解锁等接口的定义及实现。具体定义为:

CACHE_ENABLE() :Cache 使能;

CACHE_DISABLE() :Cache 关闭;

CACHE_INVALIDATE() :对整个 Cache 执行失效操作;

此外,同类功能服务接口还有 CACHE_WRITE_MODE()、CACHE_LOCK()、CACHE_UNLOCK 等。

2)　Cache 行控制

以下接口对 Cache 行的各项属性进行操作。

LCACHE_ALLOCATE(src ,size) :将 Cache 行分配给指定内存区,可用于内存块的复制;

LCACHE_FLUSH(src ,size) :在脏数据行写回内存后失效指定范围的 Cache 行;

LCACHE_INVALIDATE(src ,size) :失效所有的 Cache 行,脏数据被失效但不写回内存。

同类功能服务接口有 LCACHE_STORE(src ,size)、LCACHE_READ_HINT(src ,size)、LCACHE_WRITE_HINT(src ,size) 以及 LCACHE_ZERO(src ,size) 等。

3)　Cache 粒度设置

以下接口对指令 Cache 和数据 Cache 的大小和粒度进行操作。

CACHE_SIZE(size) :定义整个 Cache 的字节大小。

CACHE_LINE_SIZE(size) :定义 Cache 行的字节大小。

CACHE_ARRAY(num) :对每组 Cache 的组相联数目进行定义。

3.4　I/O 操作接口定义

I/O 操作接口为设备进行访问和控制时需要访问 I/O 寄存器提供用于读写操作的功能服务接口,其接口定义及功能见表 3 所示。

表 3　ESP-HAL I/O 操作服务接口

I/O 操作接口定义	功　能　描　述
IO_REGISTER	该结构用于保存 I/O 寄存器的地址
IO_READ(reg ,data ,type)	从 reg 指定的 I/O 寄存器中读取 type 类型的值并保存到参数 data 中
IO_READBRFF(reg ,buf ,num ,type)	从 reg 指定的 I/O 寄存器连续读取 num 个 type 类型的值并存放到 buf 指定的空间
IO_WRITE(reg ,data ,type)	按照 type 类型的数据长度将参数 data 写到 reg 指定的 I/O 寄存器中
IO_WRITEBUF(reg ,buf ,num ,type)	以 type 类型的数据长度为单位将 buf 指定空间的值写入连续的 num 个寄存器

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

3.5 中断处理描述

中断处理是操作系统内核的重要组成部分。操作系统内核内通常包含中断描述表,该表定义了其支持的全部中断向量,系统内核根据中断描述表的信息和中断源就可以准确地转入相应地中断服务处理程序[5]。为了最大程度地实现操作系统内核的硬件无关性,ESP-HAL 的中断管理需要为系统内核提供中断向量表管理、中断控制以及中断信息传递等服务接口。

1) 中断向量管理

该部分提供的接口有 INT_VECTOR(), INT_ISRMIN(), INT_ISRMAX(), INT_ISRCOUNT(),分别定义了中断向量表、系统中断号的取值范围以及系统中断源的数量。

2) 中断控制

ESP-HAL 中定义了中断控制服务接口,用于对具有优先设控制能力的中断控制器进行控制,其部分接口的定义如表 4 所示。

3) 中断向量 ISR 管理

该部分提供的接口支持对中断和与它相应的服务程序 ISR 进行连接、解除等操作,其接口定义如表 5 所示。

表 4 中断控制服务接口

中断控制接口定义	功能描述
INT_ENABLE()	允许所有中断发生
INT_DISABLE()	禁止所有中断发生
INT_QUERY(state)	将中断屏蔽状态保存到参数 state
INT_MASK(vec)	屏蔽特定的中断
INT_UNMASK(vec)	允许特定的中断
INT_SETPRI(vec,pri)	为特定中断设置中断优先级

表 5 中断向量管理服务接口

ISR 管理接口定义	功 能 描 述
ISR_DISPATCH(vec,isr,arg)	为 vec 指定的中断向量分配 isr 对应的中断处理程序
ISR_DELETE(vec,isr)	将中断服务程序 isr 与 vec 指定的中断向量分离
ISR_SET(vec,isr,old)	将 isr 指定的中断处理程序替换中断向量 vec 原先的 isr
ISR_GET(vec,isr)	读取中断向量 vec 对应的中断处理程序,传递到参数 isr

3.6 系统初始化

嵌入式系统的初始化过程是一个同时包括硬件初始化和软件(主要是嵌入式操作系统及系统软件模块)初始化的过程,而嵌入式操作系统启动以前的初始化操作是硬件抽象层的主要功能之一。由于嵌入式系统具有硬件环境多样性特点,使得不同的系统初始化所涉及的内容各不相同,复杂程度也有较大差异[6]。以针对 ARM7 系列 S3C4510B 处理器构建的嵌入式系统开发平台为例,启动过程主要分为 2 个阶段。

1) 第一阶段

该阶段与硬件环境紧密相关,由汇编语言实现。

(1) 硬件设备初始化 系统上电后,复位电路产生一个复位异常,使程序从 0x00000000 位置开始执行,即执行复位异常的处理程序。该处理程序直接跳转到 ESP-HAL 中提供启动功能的处理程序处开始执行。禁止 MMU,关闭数据 Cache 和指令 Cache(Cache 是相对于 ARM7 系列 CPU 来说,对于其他架构的 CPU,此处的操作可能不同)。初始化 SDRAM。在正确初始化 SDRAM 之后,为加载第二阶段的代码准备 RAM 空间。

(2) 设置堆栈 堆栈指针的设置是为了执行 C 语言代码做好准备,通常将 SP 的值放置在 RAM 空间的最顶端。堆栈设置正确与否是后阶段初始化能否顺利进行的关键。

(3) 搬移异常向量表 在堆栈设置结束之后,ESP-HAL 将那个异常向量表从 ROM 中搬移到 RAM 中,即跳转到阶段 2 的 C 代码入口,并且用各个异常填充异常处理程序。

2) 第二阶段

该阶段由 C 语言实现剩余的初始化和内核引导工作。

(1) 初始化本阶段使用的硬件设备 进行特定硬件环境的平台初始化,移植到 S3C4510B 上的 ESP-HAL

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

主要进行 I/O 接口、I2C、PCI 等功能部件的初始化操作。

(2) 读取操作系统内核映象和根文件系统映象　将操作系统内核映象和根文件系统从 Flash 上读到 RAM 空间中。

(3) 调用内核　如果内核映象是压缩文件则进行解压缩,然后为内核设置启动参数,这样就可以将剩余的初始化工作转交由内核进行处理。

4　小结

本文针对现有硬件抽象层设计上存在的不足提出了适用于嵌入式领域的硬件抽象层 ESP-HAL 的设计方法,并将其作为嵌入式系统开发平台中的一个层次予以实现。基于该平台的应用实践证明,ESP-HAL 的引入有效地提高了嵌入式系统的可移植性。

参考文献:

[1] Yoo Sungjoo, Jerraya A A. Introduction to Hardware Abstraction Layers for SoC [C]// Design, Automation and Test in Europe Conference and Exhibition, 2003. Newyork, US: IEEE, 2003: 336-337.

[2] Microsoft. Hardware Abstraction Layer for Microsoft's Windows NT Operating System [R]. US: Microsoft Corporation, 1996.

[3] Zhang Guoyin, Chen Luyuan, Yao Aihong. Study and Comparison of the RTHAL-Based and ADEOS-Based RTAI Real-time Solutions for Linux [C]// Ni Jun, Dongarra J. First International Multi-Symposiums on Computer and Computational Sciences (Vol.2). Hangzhou, China: IEEE, 2006: 771-775.

[4] 郭静寰, 孟祥迪, 郭丽虹, 等. Windows NT 硬件抽象层 HAL 功能分析 [J]. 计算机应用, 2002, 22(7): 86-88.

[5] 罗蕾. 嵌入式实时操作系统及应用开发 [M]. 北京: 北京航空航天大学出版社, 2005.

[6] 马文学, 朱名日, 程小辉. 嵌入式系统中 BootLoader 的设计与实现 [J]. 计算机工程, 2005, 31(7): 96-98.

Design and Realization of Embedded System Developing Platform Hardware Abstraction Layer

ZHANG Le, SHAO Feng-jing, SUN Ren-cheng

(College of Information Engineering, Qingdao University, Qingdao 266071, China)

Abstract: In order to resolve the problem of the hardware environment differences in building embedded system developing platform, this paper proposed the design solution and its realization method of a high portability Hardware Abstraction Layer (HAL) and then give the specification of its realization. HAL gives underlying layer's encapsulates specification and offers operating interface to upper layers' software design. HAL's introduce shields the differences of underlying hardware, improves embedded system developing platform's openness and also shortens the development period of the embedded products effectively.

Key words: embedded system developing platform; HAL; portability; HW/SW co-design

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net