一个基于Bash的轻量级构建系统的设计与实现

白 云 喻 莉 谢长生

(华中科技大学计算机系 武汉光电国家实验室 武汉 430074)

摘 要 基于在Linux各发行版本中广泛支持的Bash脚本语言,设计并实现了一个轻量级构建系统,用以管理构建过程中的各种复杂性要素,从而在多个不同硬件平台上实现嵌入式Linux图形界面操作系统的深度定制。通过该系统的独特设计和简洁实现,全新的轻量级构建系统具有对环境依赖度小、深度定制更为便捷、持续开发更为灵活高效等特点。

关键词 嵌入式Linux,构建系统,交叉编译

中图法分类号 TN911.7 **文献标识码** A

Design and Implemetation of a Lightweight Building Framework Based on Bash

BAI Yun YU Li XIE Chang-sheng

(Wuhan National Laboratory for Optoelectronics, Huazhong Science & Technology University, Wuhan 430074, China)

Abstract A lightweight building framework was designed and implemented based on Bash script language widely supported in various distribution of Linux, for the purpose of managing the complexity factors in building operation system, thus deeply customizing embedded Linux operation system with graphics user interface upon various hardware platform. Based on these unique design and simplification, the resulting system can be less environmental dependent, more efficient for continuous development.

Keywords Embedded Linux, Building framework, Cross compile

1 概述

随着移动计算(Mobile Computing)的普及,嵌入式系统(Embedded System)已经发生了深刻的变化。嵌入式处理器的性能已经迈过多媒体PC的门槛,使得嵌入式系统也可以具有现代的高级图形交互界面、网络文件系统、高性能音频、视频处理等功能,与PC机越来越相似。这类新型的嵌入式系统被称为SBC(Single Board Computer,单板电脑)[1]。

在SBC硬件平台上,可以基于Linux操作系统内核构建一个带有图形界面的操作系统,同时根据具体应用的需求,配套定制应用软件的开发和运行平台。典型的软件运行平台基本内容包括:图形用户操作界面、鼠标键盘或触摸屏的支持、硬件即插即用、中文显示与输入、TTS语音播报引擎、有线和无线网络的访问管理以及基于Qt的应用软件开发和运行环境等内容。

与以往操作系统被列为整体系统的一个采购项的做法有所不同,自主构建的操作系统中包括内核在内的绝大多数软件包均可以进行深度定制。定制操作系统不仅可以降低采购成本,还可以更为灵活高效地对已有功能进行扩展,提高内在品牌价值。例如,开机时显示产品LOGO、基于硬件产品标识的加密,或是当用户插入USB上网卡时自动拨号联网等,这些定制特性在自主操作系统中所需的开发量更小,同时设计的自由度更大。

以Ubuntu、Debian为代表,许多PC上的Linux操作系统的发行版本都通过网站的形式提供了详细构建方法,因此理论上任何人都可以利用这些知识,从零开始构建一个操作系统。但实际上,搭建一个操作系统的复杂度仍然是比较高的。这是因为软件环境是由许多相互依赖的软件包构成的,当软件包的数量增加到一定级别时,这种依赖关系会变得非常复杂;其次,深度定制时将会针对不同的需求对个别软件包进行合理的修改,而这种配置操作有时会改变依赖关系,使其更加复杂;另外,为了完成一些特定的功能,可能修改部分源码,在修改过程中可能对源码进行反复调试,大型的构建系统中这种调试比较难以实现。

因此,软件平台的构建工作本身就需要一套软件工具来辅助完成,本文称这一工具为构建系统(Building Framework)。

在总结现有构建系统设计要点的基础上,本文设计并实现了一个轻量级的构建系统。该系统重点针对嵌入式系统中的实际应用需要,面向快速定制和持续集成,进行了创新性的设计,更加简洁高效和易于掌握。本文的工作已经在TI的OMAP3530和Freescale的i.MX.536两款ARM处理器平台上进行了验证,分别搭建了基于Qt-QWS和基于X11/Gtk的多个不同规格的定制系统。

本文受国家自然科学基金(60972016),湖北省杰出青年基金(2009CDA150)资助。

白 云(1977—),男,博士后,主要研究方向为无线通信系统、网络建模等;喻 莉(1970—),女,教授,博士生导师,主要研究方向为计算机网络与多媒体通信;谢长生(1957—),男,教授,博士生导师,主要研究方向为计算机新型存储体系结构、多媒体信息存储。

2 现有系统

2.1 已有系统

OpenEmbedded[3]和 lfs[4]是应用较为广泛的、面向嵌入式系统的大型构建系统。

其中,OpenEmbedded 是一种开源的面向嵌入式的构建系统,它基于 bitbaker 脚本解释器,可以对绝大多数开源软件包进行构建。基于 OpenEmbedded 开发的嵌入式操作系统包括 Ångström[5]、Openmoko 和 SlugOS 等,这些操作系统基本具备现代图形界面操作系统所必须的大部分功能,区别只是运行在嵌入式平台之上而已。

软件包之间的"依赖性"非常适合用网站的形式来表现,lfs 就是其中之一。lfs 是 linux from scratch 的简称,即"从零开始构建 Linux"。lfs 的各种版本包括 clfs (cross-compile lfs)、blfs (beyond lfs)以及 cblfs、alfs 等等。其中 cblfs 就是关注于建立图形化界面的嵌入式 Linux 操作系统的,lfs 同时以网站、源码和书籍等形式提供构建系统。

Freescale 公司所使用的构建系统 ltib 就是基于 lfs 的,而 TI 公司的演示系统则使用基于 OpenEmbedded 构建的 Ångström。

除此之外,在非嵌入式平台上的各种主流 Linux 操作系统也正在向嵌入式平台上迁移。例如各种基于 Linux 的 PC 机操作系统包括 Ubuntu、Mint、FreeBSD 或 Debian 等。这些系统本身都是一个构建系统的产物,在对应的网站上,可以查询到每一个软件包的构建方法及其相关的软件包名称。

2.2 构建系统的定义

即使是构建小型的实用系统,构建系统管理的软件包数量也通常能达到 100 个左右,对于较为完善的构建系统,这个数字可能上升到几千甚至上万,这时仅仅考虑软件包之间的依赖关系就已经非常复杂了;一些大型的构建系统的构建时间可能达到 8 小时甚至几天,因此对于系统的稳定性和可靠性也有较高要求。

本文将构建系统(build framework)定义为对大量的源码包(source package)进行编译、管理和部署的系统,这 3 部分功能分别包括:

- 可配置的选择性编译。对每个源码包,基于一定的预定义规则进行配置和定制,并用相应的编译器进行编译。
- 源码包管理。首先,管理源码包的内容;其次,管理编译生成的内容,包括目依赖关系,从而自动组织源码包编译的先后顺序标码(Release 或 Debug 版本)、说明文档、头文件、动态或静态链接库、配置文件、资源文件等,针对总体需求进行选择。
- 部署。即对编译生成的内容进行选择性的发布,并设计合理的目录结构,添加一定的系统配置文件和正常运行所必须的脚本。

2.3 本文系统的特点

现有的构建系统大多是大而全的半商业系统,会带来下列问题:首先,默认只能针对固定的几款硬件平台,硬件选型受限;其次,系统的复杂度较高,不利于灵活高效地进行扩展;另外,需要额外的脚本解释工具支持,不具有更为通用的扩展性;最后,系统编译耗时较长,不适应于嵌入式平台的特点。

为了对自有产品实现完全的掌控和高效的定制,本文设计了一个全新的构建系统,包含如下的设计目标:

(1)易于学习和使用。即除了编译软件包所必须的知识以外,所需要学习的额外内容较少,这要求系统的复杂度较低。

(2)易于扩充和裁剪。必须假定构建系统是不断发展的,因此必须便于针对新的需求进行修改。

(3)系统在引入新的软件包进行编译和配置时,必须非常便于调试。

(4)应该有阶段性的输出目标,可以通过产品名称来标识一个阶段成果。

(5)在构建交叉编译环境时,充分利用宿主机已有的丰富功能。

基于上述设计要点,选择基于 Bash 实现该构建系统。这是因为:首先,Bash 是最为广泛支持的脚本解释器;其次,解释型语言比编译型语言有更好的需求变动适应性;最后,Bash 本身就是 linux 操作系统的命令行运行环境,这样在搭建交叉编译环境的时候,可以充分利用宿主机的功能,减少一些不必要的工作,对于一些有着特殊构建方法的软件包,也有足够的可适应性。

3 基于 BASH 的构建系统

3.1 构建方法共性机制

虽然构建方法是一个构建系统中的重要组成部分,但是对每一个软件包而言,其构建方法是相对固定的,大量参考文献[4-6]详尽地进行了描述。因此,本文不讨论具体的构建方法,仅描述一般的共性机制。

3.1.1 交叉编译

面向嵌入式的构建系统常常需要进行交叉编译(Cross Compile)。交叉编译也是一种编译(compile)形式,即从源代码(source code)编译生成目标码(object code),但是在交叉编译的情况下,编译器与目标码所处的"运行环境"可能部分或者完全不相同。

参照 GNU 的命名规范,"运行环境"可以被定义为一个唯一的字符串标识,即将每一种要素按一定顺序描述出来,例如:arm-none-linux-gnueabi。在交叉编译中有 3 种"运行环境"的概念,分别是:编译机(build)、主机(host)和目标机(target),build,host 和 target 3 个参数合起来被称为交叉编译三元组(triplet)。

3.1.2 GNU Automake

对于一个源码包而言,需要一套工具对其中的源代码进行管理和组织编译,这种工具可以统称为编译系统(build system)[2],较为熟知的编译系统包括 gnu make、ant、CMake 等等。

在 Linux 平台上的绝大多数软件包使用传统的 makefile 模式进行编译。makefile 遵循 GUNMakefile 规范,使用时最为灵活,但缺点是学习周期较长,同时生成的 makefile 与软件环境密切相关,不支持跨平台。使用 makefile 进行软件包编译一般规律是指定一个目标(target),并调用 GNU 的 make 命令进行编译。

对大型项目编写 makefile,即使是熟练的工程人员,也是一项挑战。另外,在一个操作系统中编写的 makefile,放到另外一个操作系统可能也必须进行大量的修改。为弥补 makefile 的这些缺点,GNU 早在 20 世纪 90 年代就开始了 auto-

make 框架的设计，目前这一框架已经被广泛使用，面向交叉编译的支持也越来越完善。

简而言之，automake 框架包含了 automake、autoconf、libtool、M4、Perl 等多个软件包，其最主要的作用就是根据非常精简的预定义脚本，生成一个 configure 可执行脚本，提供项目可选的配置属性，并通过 configure 脚本自动生成 makefile。同时，由于软件包之间往往不是孤立的，如何在编译时确定所依赖的其它软件包的安装位置？较为通用的方法是利 pkg-config 命令，该命令也是 GNU Automake 规范中的内容之一。

少数例外的软件包不使用 Automake 体系，比如著名的 QT 软件包，其使用了自制的 qmake 生成 makefile，一些基于 ibus 的输入法使用了 CMake 来管理源码资源，还有一些使用 PHP、TCL、Python 等脚本解释器进行辅助编译。

3.1.3 内存文件系统

绝大多数的源码包具有一些相同的特点，比如单个文件不大，但小文件的数量特别多，加起来的总量又一般不会超过几百 MB，一旦编译完成，这些源代码文件一般不再需要。

这些特点非常适合使用内存文件系统，而不是一般的磁盘文件系统。因此，本文一般会选择使用 linux tmpfs 内存文件系统作为可挥发(volatile)内容的暂存位置。

3.1.4 失败退出与日志记录

编译一个源码包时，一般情况下如果任何一个步骤异常，后续步骤就失去了继续下去的意义，必须终止构建。同时，需要快速定位故障点。因此，系统实现的通用机制包括：

- 失败退出。执行传入的 bash 命令，等待结束，获得进程结束后的返回值，在发现进程失败时退出。
- 暂存日志和告警。由于大多数软件包在编译时会输出大量的信息，这些信息在构建失败时非常有助于定位问题，但正常情况下，这些调试输出不仅拖慢构建进程，同时也会使正常的进度报告信息被完全淹没。因此利用 linux 的重定向功能，将 stdio 和 stderr 上的输出分别保存到内存文件系统的 log 和 warn 文件，并在成功编译一个软件包后删除这两个文件。

3.1.5 源码解压与补丁

软件包是以压缩包的形式呈现的，这些压缩包的压缩算法不尽相同。本文假定压缩包使用 tar 压制，压缩算法通过后缀名自动识别，可以是 gz、bz2 或者 xz。源码解压后生成一个单一目录，其目录名与压缩包的名称(去除. tar. xx 后缀后)一致。这一假定符合绝大多数源码包的实际情况，对少数特殊命名的源码包，可以人为地进行调整。

源码包常常需要在一个基础版本上进行微调，即所谓的打补丁(patch)操作。补丁实际上也是源码的一种，其重要性不亚于源码本身。本系统假定补丁是使用 diff 工具生成的，打补丁操作通过 patch 工具来完成。

源码解压、打补丁这一过程被包装为 prepare 操作，即将源码包解压到指定的内存文件系统目录，并对解压后的源码打补丁。

3.1.6 交叉编译的一般步骤

在上文描述的实现基础之上，设计实现了 build_native 脚本一步完成解压、补丁、配置、编译、安装的全部操作，并可进行少量的微调，包括：

- 更换源码包名称，用于更换同一源码的不同版本；
- 更换源码包的编译配置命令；
- 不要 prepare，即假定源码包已经解压完成；
- 临时编译指定的 target，这在编译一个较大源码包中特定子库时非常实用；
- 指定在源码目录平级的临时目录中还是在源码目录内编译。前者是一些源码包，比如 linux 内核、glibc 等，为了避免污染源码目录，而建议采用的编译方式。但如果已经完成部分编译，仅需要再编译部分源码包时，后者要快得多。
- 是否通过 make check 编译运行系统测试以验证可用性。在大多数交叉编译的环境中，目标码在编译机中无法执行，因此没有办法进行系统测试。

3.2 依赖管理

软件包之间存在着依赖关系，比如图 1 所示的是准备构建 X11 所需要的最为基础的几个软件包之间的依赖关系示意图。在实际情况中，软件包的依赖关系远比该图复杂。

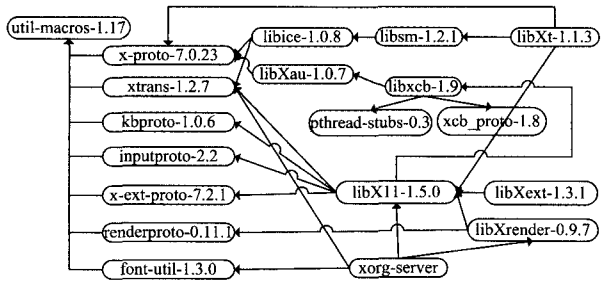

图 1 部分 X11 软件包之间的依赖关系图

本文设计的构建系统依赖关系管理机制如下：

首先，将整个构建进程定义为若干任务(TASK)，每个任务对应为编译一个具体的软件包(PACKAGE)，为方便调试，使用一个中文字符串作为任务的唯一标识。

其次，将 TASK 分为 build 和 compile 两个阶段，分别对应函数名为 build_taskname() 和 compile_taskname() 的 BASH 函数。只有在 compile_xxx()中才会真正开始编译一个软件包，同时规定 compile_taskname ()只能由相应的 build_taskname ()调用在 prep 阶段。对于本软件包所依赖的每一个软件包，依次调用相应的 build_xxx()函数，然后再使用工具函数 run_task 启动自身的构建。例如，编译处理字体文件的 freetype 软件包时，需要用到处理压缩算法的 zlib 软件包，其 prep 阶段代码如图 2 所示。

```
build_libfreetype()
{
    build_libz
    run_task "构建 $ FREETYPEFILE" "compile_libfreetype"
}
```

图 2 交叉编译的标准步骤

接下来，run_task 在启动相应的 compile_taskname()函数以前，会试图在一个内存文件中加入一行该任务的标识，这一内存文件在启动构建系统的时候会被清空。run_task 脚本如果发现该标识已经存在，就会直接返回，而不会启动编译工作。

同时，为保证多次调试时不需要重复编译已经调试完成的软件包，complie_taskname 在启动前会首先检查缓存，如果存在同名的压缩包，说明已经成功编译过，将不再重新编译。

这一设计保证了：1)所有的任务只会运行一遍；2)依赖的

任务一定先于本任务自身运行。在这两点的保障下，构建系统会自动安排编译执行顺序。

不过，在这样的设计下，依赖环路仍然是一个问题。首先，从逻辑上分析，依赖环路是一种错误，必须与实际中多次构建同一个软件包的情况区别开。例如，在搭建交叉编译环境时，首先需要编译基础的 gcc 软件包，使其成为一个 C 语言编译器，接着编译 glibc(或 uclibc 等其他等价实现)实现标准 C 运行时库，然后再次编译 gcc，使其支持 C++编译。需要注意的是，这样两次编译时使用的配置参数是不同的，compile_xxx()脚本中的内容也必然不同，因此应该视为不同的构建任务。因此，无论软件包之间的依赖关系多么复杂，其关系图中都不可能出现环路，环路的存在一定是由于构建方法存在着错误，应该在构建系统启动之初检测出来。

本系统只进行环路检测。环路检测可通过设置使 run_task 不调用实际的编译操作 compile_xxx，再重复一次整体的构建，这样的“伪”构建应该很快完成。在存在回路时，“伪”构建会出现死循环，在一定时间后(比如 10 秒钟后)伪构建没有结束，说明存在依赖关系环路，即可中止运行。这时，run_task 负责写入的任务标识文件列表中最后一个任务标识，就是出现依赖环路的故障点。

大多数大型构建系统是通过配置文件管理依赖关系的，而本文中的 prep 阶段实际上完成了相同的工作。在开发的阶段，将依赖关系配置和实际编译的代码放在一起更方便调试；在产品定型后，也可以将这些部分分别保存到不同的 BASH 脚本中去。省略专门的配置文件也是为了减少需求变更时的修改点。

3.3 产品定义

现实中构建系统的开发是没有终点的，因为操作系统会针对新特性不断完善和升级。但反映构建系统的阶段性成果也是一个必需的要求，这一点可以通过产品发布的形式完成。与产品发布相关的概念大致有如下几个，在本构建系统中均体现为不同的 BASH 环境变量：

• 产品(PRODUCT)。不同的产品名称可能采用不同的硬件体系结构、不同的 Linux 内核、不同的软件包，因此 PRODUCT 决定了整个构建系统需要进行什么样的构建操作。

• 配置(CONFIG)。希望目标系统具有什么特性，例如：X11 窗口系统(CONFIG_X11)、需要音频处理 (CONFIG_SOUND)。构建系统会根据这些配置，选择不同的软件包组合。

• 能力(CAPABILITY)。受 PRODUCT 选择的硬件平台限制，所具有的能力选项。比如具有硬件图形协处理器(CAP_GPU)，可以编译图形加速的相关功能，否则只能选择软件实现的驱动。

• 源码包(PACKAGE)。对应一个具体的源码包，比如 PACKAGE_XORG= xorg-server-1.12.4，包对应的属性有版本号、编译方法、编译参数、编译步骤等，均通过一个 BASH 函数实现。但一般而言，即使同一软件包的不同版本，其编译方法也可能完全不一样，这时就必须视为不同的 PACKAGE 来处理。

• 发布包(DIST)。一个源码包(PACKAGE)可以对应多个发布包(DIST)，最常见的是从同一份源码产生运行态和开发态的两个发布包。前者仅包含可执行文件、链接库及必须的配置文件等；而后者还必须包括开发所需的头文件和构建所需的辅助脚本等。

本构建系统仅将 PRODUCT 做为可选配置变量，所有配置均通过 BASH 函数得以体现，如图 3 所示。

```
#). /build_all. sh fsl_ecs
#). /build_all. sh ti_ecs
```

图 3 分别构建基于 freescale 和 ti 两款芯片的 ecs 产品平台

这样会分别执行 construct_fsl_ecs()和 construct_ti_ecs()函数，设置相应的 PLATFORM 和 CAPABILITY 变量，然后根据 CONFIG 不同调用不同的 BASH 函数，例如 CONFIG_X11 则调用 construct_x11()函数，在 constrct_x11 函数中再具体地组装各个软件包，比如 build_xorg_server()、build_gtk2()、bulid_qt_x11()等。在前面介绍的依赖管理机制作用下，只需要指定最主要的几个软件包即可，其它的依赖软件包会自动执行。

软件包的选择是依赖于 PRODUCT 和 CAPABILITY 变量的(后者实际上是一组变量)，而 PRODUCT 的定义实际上包含了产品(软件范畴)和平台(硬件范畴)两重概念。比如，内核的编译 build_kernel()会根据 PRODUCT 变量选择相应厂商提供的源码包，可以自行决定是分别实现 build_kernel_fsl()、build_kernel_ti()，还是在一个 build_kernel 中通过多个 if-else 来实现。考虑到不同产品中有若干功能是重叠的，本文实际采用了后面一种方式。

3.4 发布助手

对于编译完成后安装的内容，本系统设计了 4 种组态：

(1)CACHE。所有文件被直接安装到 $CACHEDIR 目录，即 make install DESTDIR= $CACHEDIR。这时发布助手将其打包为一个压缩文件，CACHE 组态的文件包含了一个软件包可能生成的所有内容。

(2)BOOT。以 u-boot 和 uImage 等负责引导为主的生成内容，这部分内容有可能需要通过直接 io 操作写到指定扇区，因此单独保存到 $BOOTIDR 目录。

(3)DIST，发布态文件。这部分文件构成了嵌入式系统实际运行时的根文件系统，可以直接以文件的形式存在，也可以打包成 rpm 或 deb 格式的安装包。由发布助手自动从 $CACHEDIR 中复制过来，但一般会去除其中的静态库(.a)、头文件(.h)、libtool 文件(.la)、man 文件(.man)及其他帮助文件，同时对可执行文件、库文件(.so)进行 strip 操作以减小发布映像的大小。

(4) SDK，开发态文件。这部分文件由发布助手从 $CACHEDIR 中直接复制过来，在编译其它软件包时，可能需要其中的头文件、静态库等内容，但实际运行时并不需要，因此不需要发布到根文件系统中以减小容量占用。同时，pkg-config、autoconf 等交叉编译工具也被指向这个目录，有助于避免交叉编译环境受到宿主机环境污染。

3.5 交叉编译环境污染

在交叉编译的时候，软件包可能读取编译机(build)的环境进行配置，从而导致错误的编译结果，这种现象被称为交叉编译环境受到污染。

这种污染的成因可能是多方面的，比如错误地读取了/usr/include 目录下的头文件，从而错误地认为目标主机具有

某种能力，而编译出错误的代码；或者，错误地读取了/usr/lib/pkgconfig 目录的.pc 文件，从而错误地认为目标主机有某些软件包，从而导致运行时发生找不到动态链接库的错误；又比如，在配置时错误地读取了/usr/share/aclocal 中的文件，从而生成了错误的 configure 脚本。

一般情况下，编译机(build)具有比主机(host)更强的能力，环境污染会误导交叉编译器使用错误的编译方式，而这种错误会一直隐藏到运行时才得以发觉，因此危害较大。

"解决交叉编译环境的污染问题"和"适当利用宿主机现有的工具"二者之间是一个矛盾的关系。大型的构建系统自行设计了一套工具(fakeroot、sandbox 等)，从根本上解决了环境污染的问题。但这样就完全不能利用宿主机现有的功能了，同时需要构建一个"完整而纯净的"交叉编译环境，对于某些小型的嵌入式应用，这一工作量反而超过了构建产品本身所需的工作量。

本文自行构建了较新版本的 pkg-config、automake、autoconf 和 libtool 等工具，同时在 SDK 目录中部署了宿主机版本的 glib 可执行文件(glib-compile-schema 也是一个编译工具)，但对其它的工具包，比如 perl 语言环境、phyton 语言环境、CMake 编译工具等，由于较少涉及而忽略。

即使如此，有些软件包仍然需要在 configure 的时候在宿主机和目标机环境之间切换，这时，可以采取如下 3 种办法临时适应其具体需求：

(1) Hide/Restore Header。将/usr/include 和/usr/local/include 这些编译器预设路径临时隐藏。

(2) Hide/Restore Autoconf。将/usr/share/aclocal 等目录临时隐藏。

(3) Hide/Restore pkgconfig。将/usr/lib/pkgconfig 和/usr/share/pkgconfig 等目录临时隐藏。

结束语 本文设计了一个基于 Bash 的自制构建系统，展示了深度定制操作系统的可能性。本文设计的构建系统具有实现机制易于学习掌握、支持增量调试与开发方便引入新增特性、支持按特性描述的配置项定义以支持不同硬件平台和不同的软件特性等特点，同时基于普遍支持的 bash 脚本环境具有最大的可适应性，并可按产品名称为标识输出阶段性的研发成果。

目前，整个构建系统的实现框架已基本定型，编译方法模块也已经初具规模，基本实现了与硬件密切结合的 QT_QWS 及 Qt_X11 两类不同规格的自主定制的软件环境。未来的工作将可以不断引入新的软件包和新的功能，同时对已有的功能进行升级和修改，使得应用系统越来越完善。

参 考 文 献

[1] Rosli A N C, Shakaff A Y M, et al. Face reader for biometric identification using Single Board Computer and GNU/Linux[C]// International Conference on Robotics, Vision, Information, and Signal Processing, 2007. Penang, Malaysia, 2007

[2] Adams B, Schutter K D, Tromp H, et al. The Evolution of the Linux Build System[C]// The Third International ERCIM Symposium on Software Evolution, 2007. 2007

[3] Lauer M. Building Embedded Linux Distributions with BitBake and OpenEmbedded[C] // Proceedings of the Free and Open Source Software Developers' European Meeting (FOSDEM). Brussels, Belgium, Feb. 2005

[4] Beekmans G. Linux From Scratch Version 6.2[M]. onlinewebli-brary. com, 2006

[5] Angstrom Web Site[OL]. http://www. angstrom-distribution. org/, 2012

[6] Yaghmour K, Masters J, Ben-Yossef G, e t al. 构建嵌入式 LINUX 系统(第 2 版)[M]. Taiwan O R 公司，译. 秦云川(改编)，中国电力出版社，2011

(上接第 3 页)

人自己最近的簇，形成大小规模不等的簇；数据转发采用簇内单跳，簇间单跳、多跳相结合的方式，距离基站较近的特殊节点直接与基站通信。通过降低网络内通信来减少网络能量消耗，均衡网络开销。实验证明，SIAUCR 算法均衡了网络能耗，能量利用率、网络寿命要优于 LEACH、CEBRCA，同时数据发送效率也获得较好效果。

参 考 文 献

[1] Patel H, Pandya N. Study and Review of Routing protocols for wireless sensor networks[J]. International Journal of Engineering, 2013, 2(1)

[2] Norouzi A, Babamir F S, Zaim A H. A novel energy efficient routing protocol in wireless sensor networks[J]. Wireless Sensor Network, 2011, 3(10): 341-350

[3] Manap Z, Ali B M, Ng C K, et al. A Review on Hierarchical Routing Protocols for Wireless Sensor Networks[J]. Wireless Personal Communications, 2013: 1-28

[4] Heinzelman W R, Chandrakasan A, Balakrishnan H. Energy-efficient communication protocol for wireless microsensor networks[C]//Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, 2000. IEEE, 2000, 2

[5] Mhatre V, Rosenberg C. Design guidelines for wireless sensor networks: communication, clustering and aggregation[J]. Ad hoc Networks, 2004, 2(1): 45-63

[6] Kuila P, Jana P K. An energy balanced distributed clustering and routing algorithm for Wireless Sensor Networks[C] // Parallel Distributed and Grid Computing (PDGC), 2012 2nd IEEE International Conference on. IEEE, 2012: 220-225

[7] Ali S A, Sevgi C. Energy Load Balancing for Fixed Clustering in Wireless Sensor Networks[C] // New Technologies, Mobility and Security (NTMS), 2012 5th International Conference on. IEEE, 2012: 1-5

[8] Heinzelman W B, Chandrakasan A P, Balakrishnan H. An application-specific protocol architecture for wireless microsensor networks[J]. IEEE Transactions on wireless communications, 2002, 1(4): 660-670

[9] Vinh T T, Quynh T N. EMRP: Energy-Aware Mesh Routing Protocol for Wireless Sensor Networks[C] // Advanced Technologies for Communications (ATC), 2012 International Conference on. IEEE, 2012: 78-82

[10] Jia Yun-jie, Liu Ming, et al. A clustering routing algorithm based on energy and distance in WSN[C] // Computer Distributed Control and Intelligent Environmental Monitoring (CDCIEM), 2012 International Conference on. IEEE, 2012: 9-12

[11] 蒋畅江，石为人，唐贤伦，等. 能量均衡的无线传感器网络非均匀分簇路由协议[J]. 软件学报，2012, 23(5): 1223-1232