BOCCN
博讯科技

JN516X开发指南

Version 1.2

北京博讯科技有限公司
www.boccn.com.cn

Table of Contents

更新历史

版本号	时间	更新内容
V1.0	2014-07-18	第一次发布
V1.1	2014-07-30	第二章增加各协议对比，测试板原理图,最小系统图；第五章增加部分外围部件操作内容 UART, DIO/DO，TickTimer, Flash，Wake Timer，ADC，EEPROM，PulseCounter, I2C，SPI 部分内容
V1.2	2014-8-29	第四章增加 MIB Traps 内容

1：基础概念

为了使我们更好的完成 JN516x 开发技术的学习，首先我们希望通过概念的介绍在大家的头脑中形成一些统一的概念，这样在我们下面的文档中提到这些概念的时候我们可以不用再进行解释。

ZigBee

“ZigBee”是一个协议的名称，这一协议基于 IEEE 802.15.4 标准，其目的是为了适用于低功耗，无线连接的监测和控制系统。这一协议标准由 ZigBee 联盟维护。在协议栈之上，ZigBee 又针对各种应用领域定义了一系列的应用子集（Application Profile），例如 Home Automation、Smart Energy，Light Link 等。

IEEE 802.15.4 标准

IEEE802.15.4 是 ZigBee 协议的底层标准，主要规范了物理层和 MAC 层的协议，其标准由国际电工学协会 IEEE 组织制定并推广。

2.4G 免费频段

免费频段，是指各个国家根据各自的实际情况，并考虑尽可能与世界其他国家规定的一致性，而划分出来的一个频段，专门用于工业，医疗以及科学研究使用的，不需申请就可以免费使用的频段。我们国家的 2.4G 频段就是这样一个频段。

PAN

Personal Area Network 的缩写，用于区别同一 Channel 中，不同的节点群组，只有属于同一个 PAN 的节点之间才能相互通讯。

WPAN

Wireless Personal Area Network 的缩写， 基于无线通讯的 PAN.

Channel

通常翻译成通道，ZigBee 所使用的频率范围从 2400MHz 到 2483.5MHz 共 16 个通道，同一个网络的设备必须位于同一个通道中。

MAC 地址 / Extended Address

MAC 地址是网络设备的一个唯一标识码，这一编码具有全球唯一性，由 IEEE 进行管理。

短地址 / Network Address

当 ZigBee 装置加入一个 PAN 中时，会由上一层父节点分配一个 16 位地址，用于网络内节点之间的标识和通讯，以减小包的大小。

Coordinator

ZigBee 网络中的一种网络设备的角色定义，用以控制整个 PAN,每一个 PAN 都必须有一个 Coordinator

Router

ZigBee 网络中的一种网络设备的角色定义，用以转发数据，延伸 ZigBee 网络的规模。

End-Device

ZigBee 网络中的一种网络设备的角色定义，作为网络的最终端节点。

JenNet-IP

由 NXP 所开发并维护的一种专门面向住宅、商业和工业等物联网应用推出的基于 802.15.4 的无线网络协议栈。

2：开发平台

2.1 硬件平台

JN516x 芯片

JN516x 是一系列超低功耗， 高性能 MCU， 集成了 IEEE802.15.4 兼容的 2.4GHz 射频收发器。 该系列芯片包含 JN5168-001, JN5164-001， JN5161-001 三个型号。 根据型号不同配置有不同的 RAM 和 Flash。 本手册如不进行特殊说明都是面向 JN5168-001 芯片， 这一芯片具有相对更大的 RAM 和 Flash， 因此支持的协议栈更加的丰富。

如果您需要了解 JN516x 芯片的管脚和更多的参数细节，您可以参考 JN516x 的 DataSheet
http://www.boccn.com.cn/getfile.aspx?id=38

JN5168-001-Mxx 模块

JN5168-001-Mxx 系列模块是基于 JN5168 芯片所开发的一系列表贴块产品。该系列模块集成了所有的射频组件和无线微控制器。采用模块进行开发可以大大的减少开发人员的工作量，缩短产品的开发周期。

这一系列的模块包含下列不同的型号

JN516x-001-M00/ JN516x-001-M03 (标准功率模块)

·可视距离<1km

·M00：板载天线(16x30mm)

·M03：uFL 连接器(16x21mm)

·发射功率：+2.5dBm

·接收器灵敏度：-95dBm

·TX 电流：15mA

·RX 电流：17.5mA

·工作电压：2.0-3.6V

JN516x-001-M05 (高功率模块)

·可视距离：<2km

·M05：uFI 连接器(16x30mm)

·发射功率：+9.5dBm

·接收器灵敏度：-96dBm

·TX 电流：35mA

·RX 电流：22mA

·工作电压：2.0-3.6V

JN516x-001-M06 (超高功率模块)

·可视距离：<6km

·M05：uFI 连接器(16x30mm)

·发射功率：+22dBm

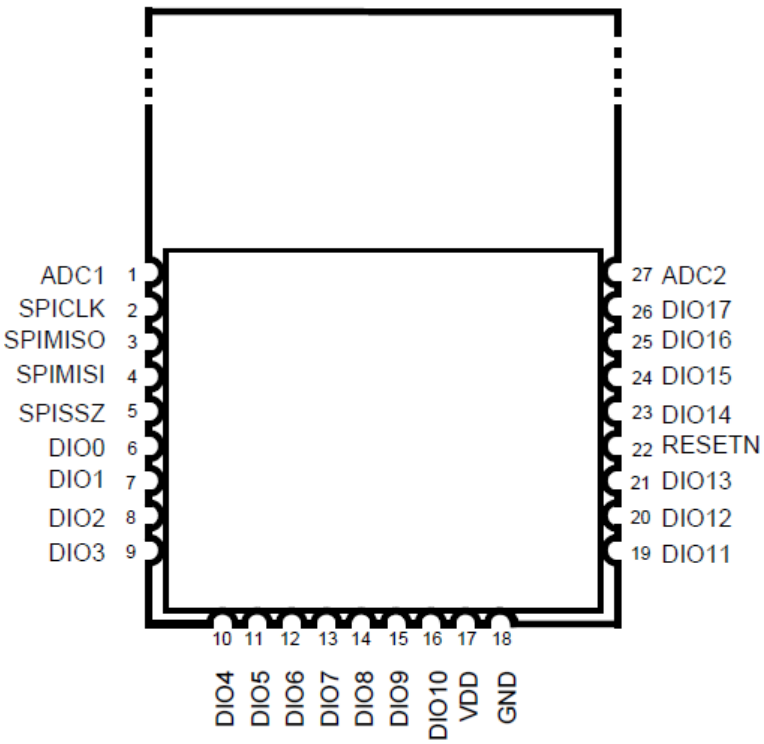
·接收器灵敏度：-100dBm

·TX 电流：175mA

·RX 电流：22mA

·工作电压：2.0-3.6V

JN5168 系列模块的针脚定义



如果您需要了解关于模块产品的更多细节，可以参考模块产品的 DataSheet

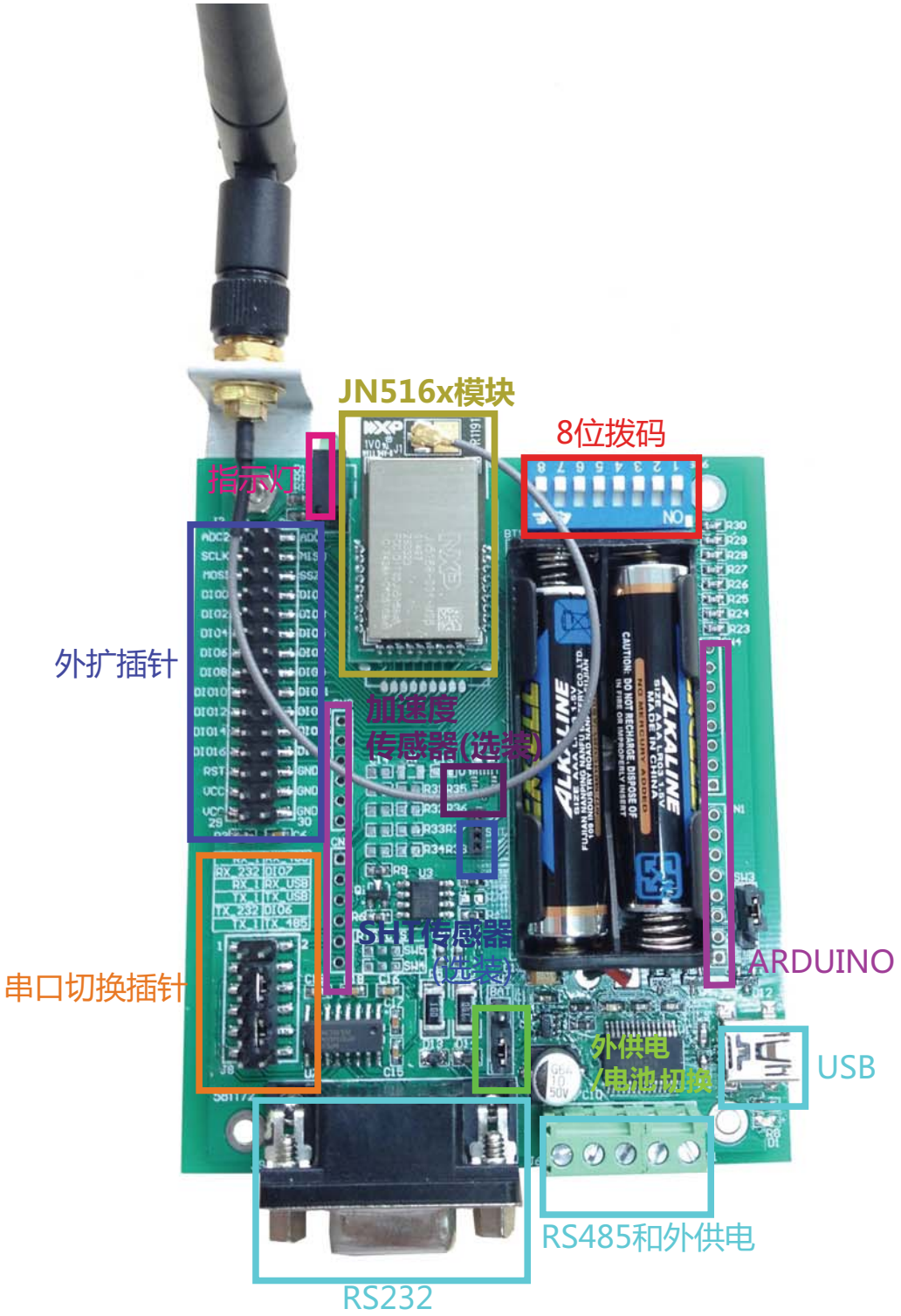
http://www.boccn.com.cn/getfile.aspx?id=141

开发包

本手册将采用北京博讯所发售的 JN516x-DK 系列开发包进行说明

JN516x-DK 开发包主要包括数个基于模块的开发板，开发板可搭配 JN516x 系列模块，具有 RS232，RS485 接口，USB 接口，SHT 系列温湿度传感器，加速度传感器，可供模拟 IO 变化的 8 位拨码，以及 4 路 ADC 等；丰富的接口以及周边硬件将缩短入门以及开发时间，节省您宝贵的时间成本。

下面是关于开发板接口的详细说明：


JN516x模块
8位拨码
指示灯
外扩插针
加速度
传感器(选装)
SHT传感器
(选装)
串口切换插针
ARDUINO
外供电
/电池切换
USB
RS485和外供电
RS232

青色的部分为接口部分电路：

USB：USB 接口可以为开发板供电，并在 PC 上虚拟为串口，可以直接调用通信；

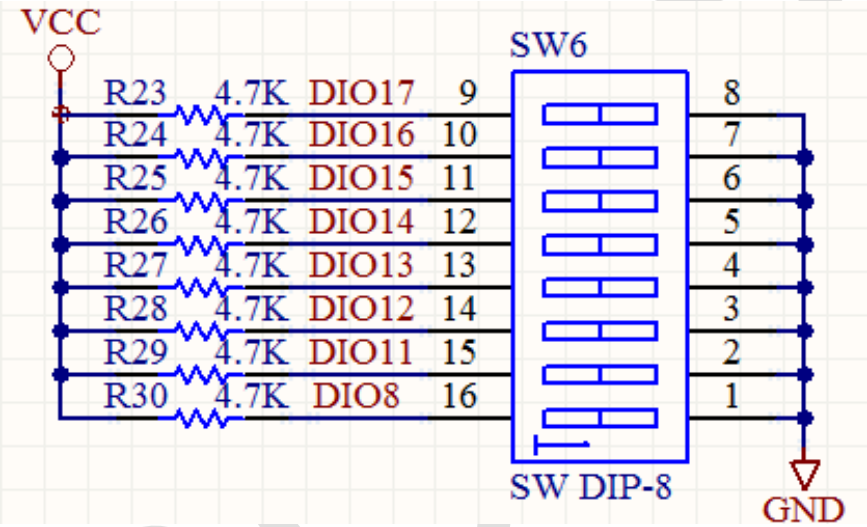
RS-232：J9 是 RS-232 接口；

RS-485：J6 是 RS-485 接口;

电源：外置电源供电接口，供电 5V，1A max；

红色的部分为拨码端子：

拨码端子非闭合状态为 IO 拉高，闭合状态为 IO 拉低；



IO 分别为：

第一位：DIO8

第二位：DIO11

第三位：DIO12

第四位：DIO13

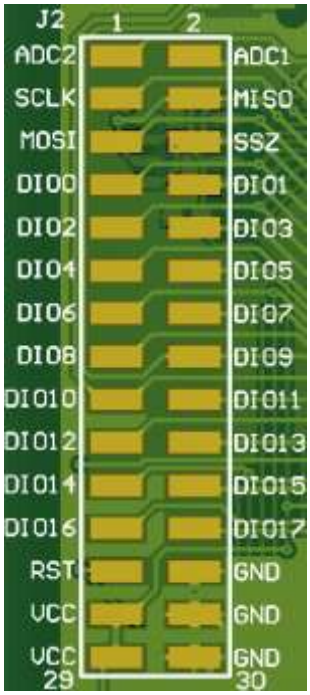
第五位：DIO14

第六位：DIO15

第七位：DIO16

第八位：DIO17

紫色的部分为外扩插针：



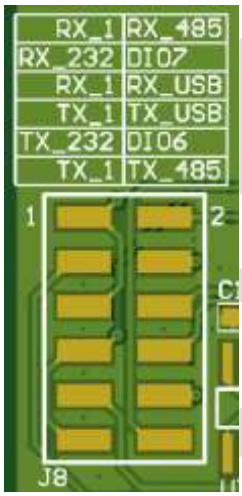
这部分将所有 JN516x 的所有焊盘都外引到插针焊盘上，方便与其他的硬件相接。

焊盘	定义	焊盘	定义
1	ADC2	2	ADC1
3	SCLK	4	MISO
5	MOSI	6	SSZ
7	DIO0	8	DIO1
9	DIO2	10	DIO3
11	DIO4	12	DIO5
13	DIO6	14	DIO7
15	DIO8	16	DIO9
17	DIO10	18	DIO11

19	DIO12	20	DIO13
21	DIO14	22	DIO15
23	DIO16	24	DIO17
25	RST	26	GND
27	VCC	28	GND
29	VCC	30	GND

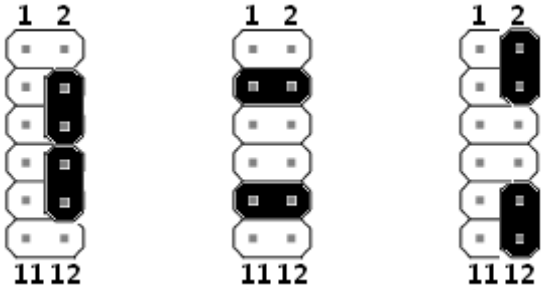
橙色的部分是串口切换插针：

通过改变插针端子的位置可分别将 JN516X 模块的 UART0 和 UART1 分别与 RS232，RS485, 和 USB-Serial 电路相接，实现串口的转接和测试；

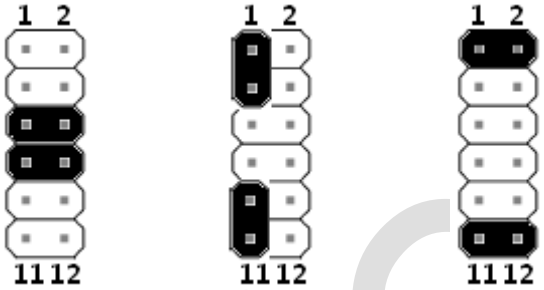


示例：

UART0 – USB　　**UART0 –RS232**　　**UART0 – RS485**

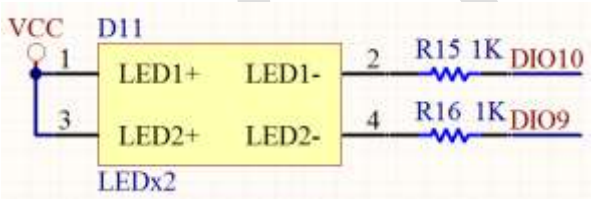


UART1 – USB　　**UART1 –RS232**　　**UART1 – RS485**



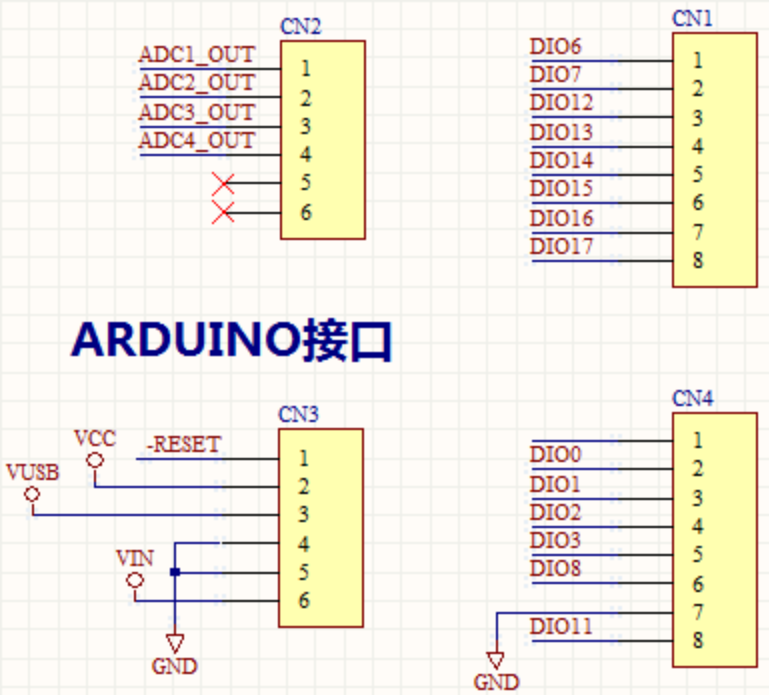
粉红色的部分是 LED 指示灯：

两个 IO 控制 LED 指示灯，分别是 DIO9(上)， DIO10(下)；

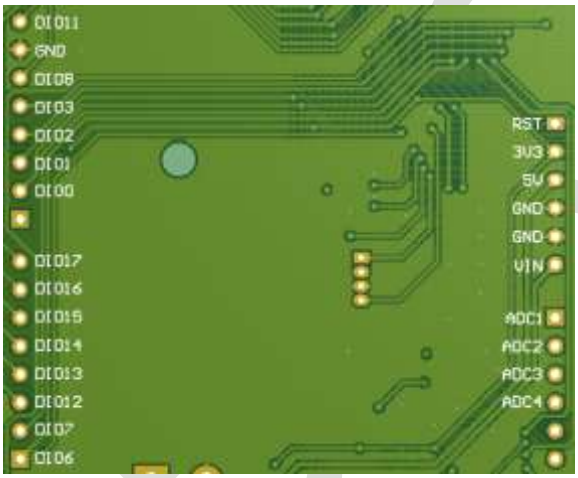


浅紫色的部分是 ARDUINO 硬件接口：

可以与开源硬件 ARDUINO 实现对接；



板子背面有定义标注，仰视图（背面）



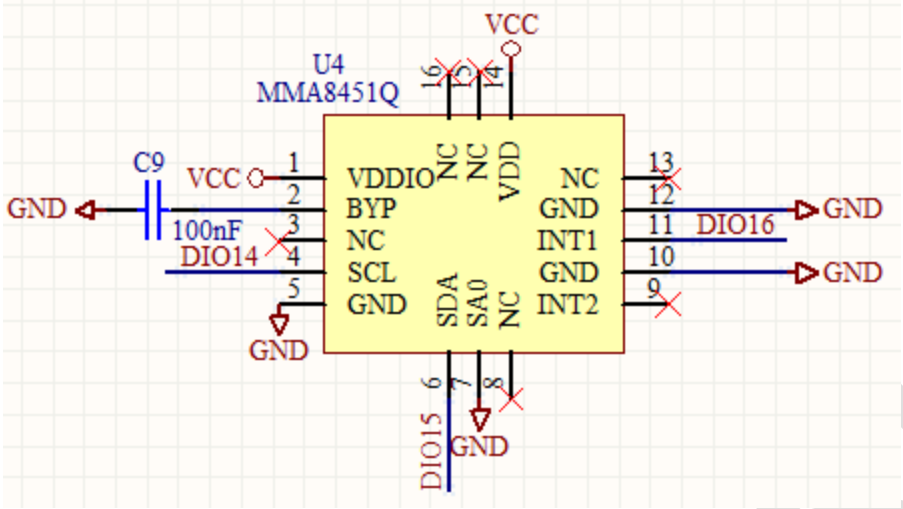
深紫色的部分是加速度传感器（选装）：

可选装加速度传感器，型号为 MMAQ8451,其中：

DIO14 为 I2C_CLK,

DIO15 为 I2C_SDA,

DIO16 为加速度中断触发；

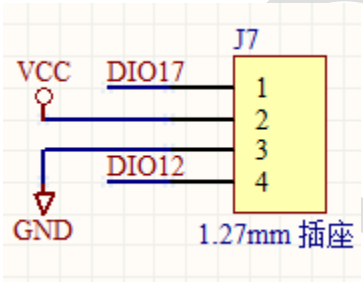


蓝色的部分是 SHT 温湿度传感器（选装）：

可选装 SHT 传感器，进行温湿度的测量，接口为类 I2C 接口，其中：

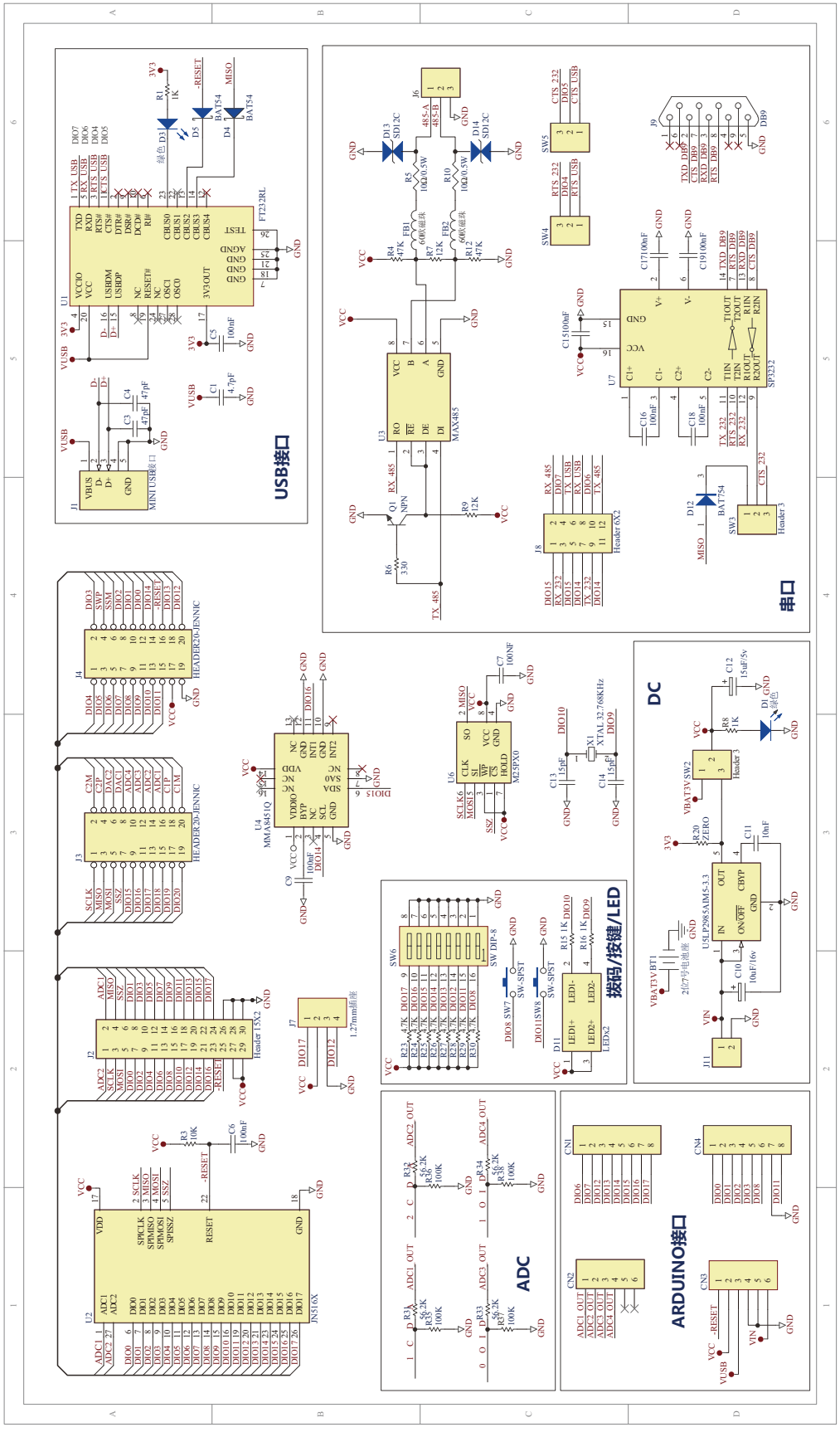
DIO17 为 CLK,

DIO12 为 SDA;

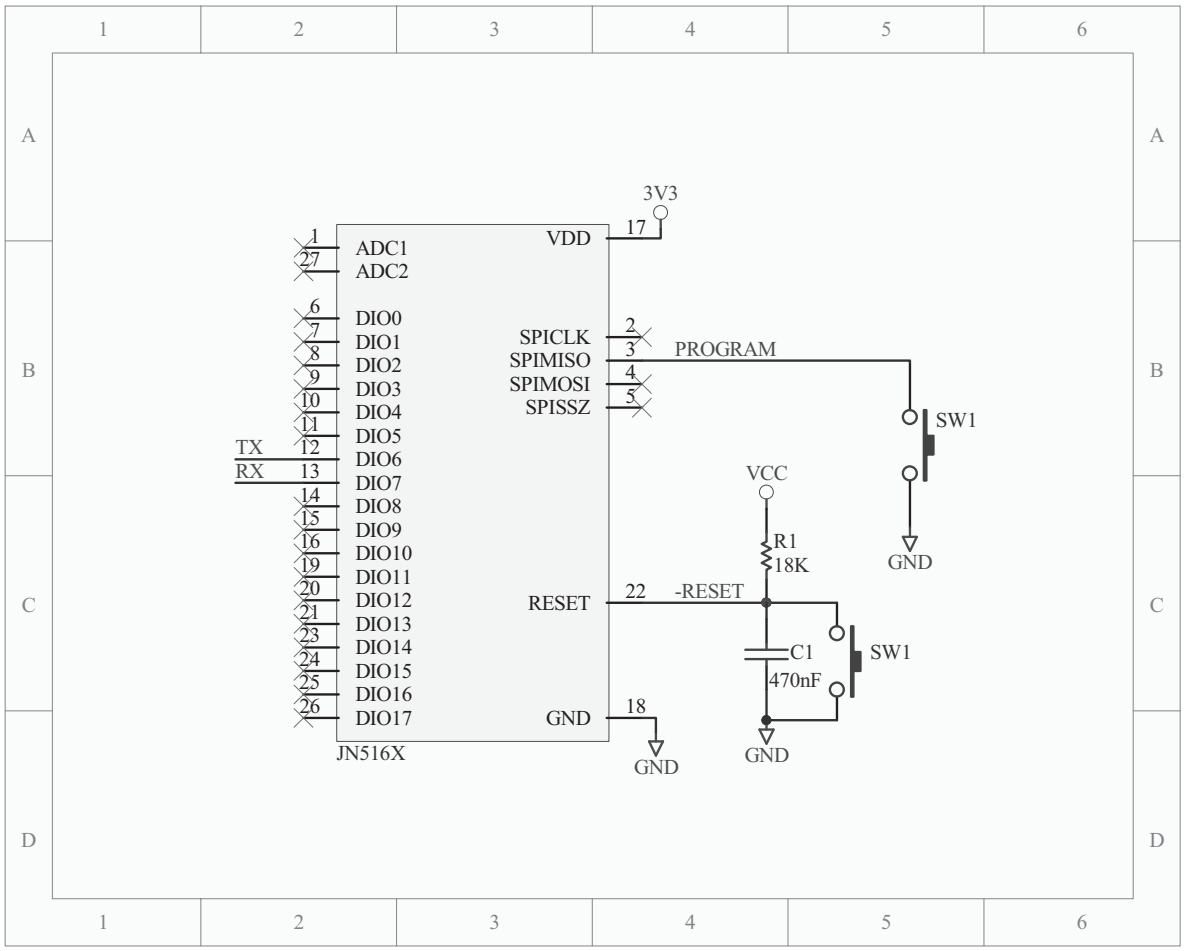


另：可以选焊 M25PX0 以及类似的 SPI 接口 Flash，扩展 Flash 容量。

开发板原理图



最小系统图



关于开发板的更多详细资料可以到北京博讯科技有限公司官网下载

http://www.boccn.com.cn/

2.2 开发环境安装与调试

JN5168 的开发工具在北京博讯科技有限公司官网都可以免费的下载。

http://www.boccn.com.cn/productThree.aspx?id=54

目前北京博讯官网提供的各种文档和工具还是比较丰富的， 但是因为种类数量众多， 所以我这里将最先需要安装下载的几个软件列出来， 其他的工具和文档您可以根据需要以后再下载安装。

JN51xx Flash Programmer-JN-SW-4007

这个是用于向芯片写入编译好的程序的写入工具。 虽然上面的开发工具包中已经包含了一个较低版本的写入工具， 但是建议还是使用最新的独立版本的 Flash Programmer (否则会出现部分模块无法读出 mac 地址， 无法写入程序的情况). 本开发手册使用的是版本号为 1.89 的 Flash Programmer

http://www.boccn.com.cn/getfile.aspx?id=248

JN51xx SDK Toolchain- JN-SW-4041

这个是用于开发面向 JN5168 应用的软件开发工具包， 包含了完整的编译工具，Eclipse 开发环境等等必备的软件。

http://www.boccn.com.cn/getfile.aspx?id=1045

JN516x IEEE802.15.4 SDK- JN-SW-4063

这是开发基于 802.15.4 协议栈应用所必须安装的 SDK. 需要在安装 JN51xx SDK Toolchain 之后安装

http://www.boccn.com.cn/getfile.aspx?id=1039

JN516x JenNet-IP SDK- JN-SW-4065

这个是开发基于 JenNet-IP 协议栈应用所必须安装的 SDK, 需要在安装 JN51xx SDK Toolchain 之后安装

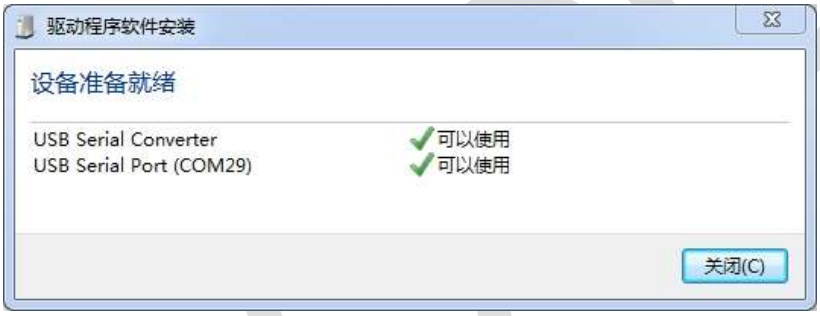
http://www.boccn.com.cn/getfile.aspx?id=266

软件的安装过程：

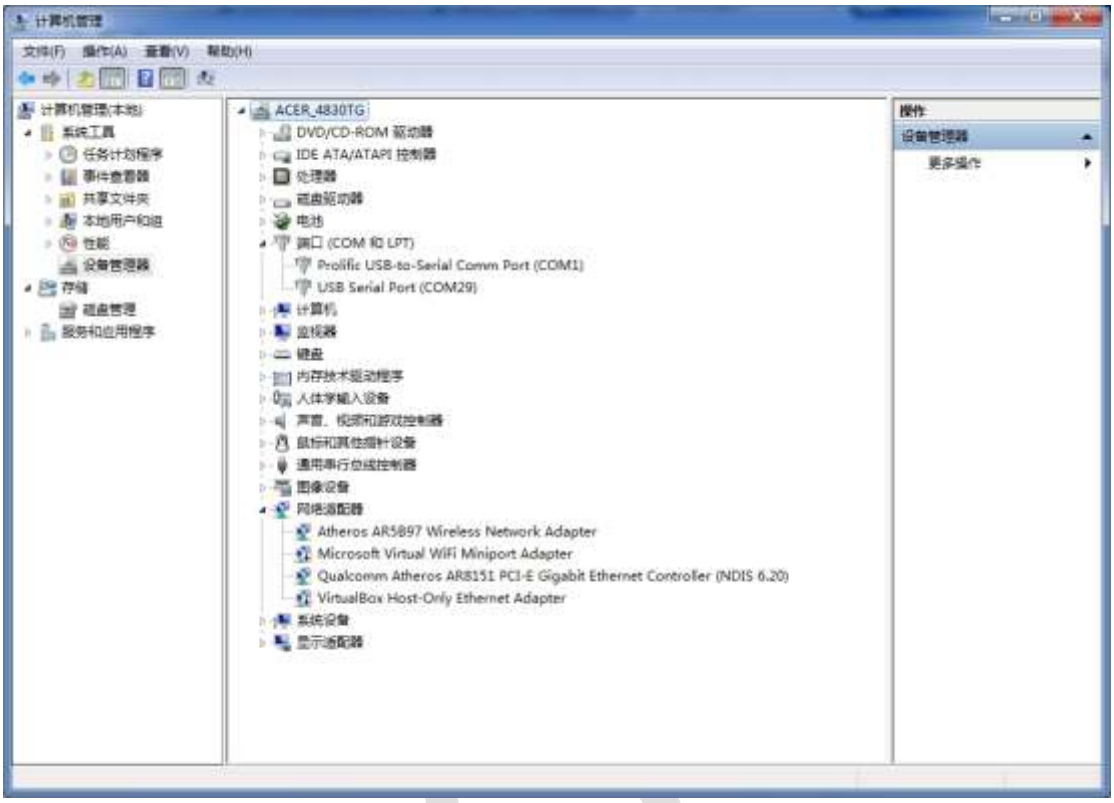
要想使用 JN51xx Flash Programmer 需要事先安装开发板的串口转 USB 驱动程序。

当开发板的 USB 接上电脑后会提示安装驱动程序，如图：

驱动安装成功：



在设备管理器的端口选项里可以看到“USB Serial Port （COMxx）”：



这时可以通过调用这个串口来对 JN516x 模块进行通信。

32-bit 驱动下载：

http://www.ftdichip.com/Drivers/CDM/CDM%202.08.30%20WHQL%20Certified.zip

64-bit 驱动下载：

http://www.ftdichip.com/Drivers/CDM/CDM%202.08.30%20WHQL%20Certified.zip

安装完成后就可以使用 Flash Programmer 通过开发板的编程口向开发板写入程序了。

开发环境安装：

下载 JN51xx SDK Toolchain 之后可以看到 JN-SW-4041.zip， 解压这个文件， 运行其中的 JN-SW-4041-SDK-Toolchain-v1.1.exe 程序。 一切都按照默认的设置进行安装。 建议不要改变默认的安装路径 C:\Jennic

ToolChain 安装完成之后解压安装

JN-SW-4063 .zip　　JN516x IEEE802.15.4 SDK

按照默认的设置安装到 ToolChain 的目录下面， 不需要修改任何设置

然后解压安装

JN-SW-4065.zip　　JN516x JenNet-IP SDK

也是按照默认的设置安装到 ToolChain 的目录下面， 不需要修改任何设置

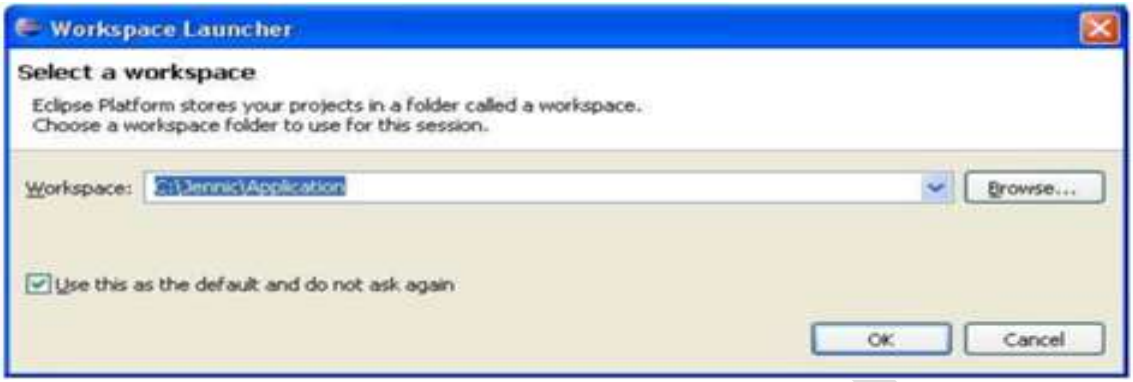
这样基本上开发环境和相应的协议栈 SDK 就安装完成了， 如果您需要开发基于各种 ZigBee 协议栈的应用， 也可以到博讯网站上下载到相应的 SDK, 或者联系我们的技术人员索要 ZigBee 相应协议的 SDK. 安装过程是大致相同的。

再次提醒大家注意， 所有的协议栈 SDK 都需要在安装了 ToolChain 之后才可以安装。

软件安装完成之后就需要配置 Eclipse 开发环境了。 面向 JN5168 的开发我们将使用 Eclipse 开发环境来进行代码的开发。 Eclipse 是一个开源的开发环境， NXP 对其进行了客制化以适合 JN516x 的应用开发。

首先从开始菜单启动 Eclipse

开始菜单 -> Programs -> Jennic -> JN-SW-404x products -> Eclipse



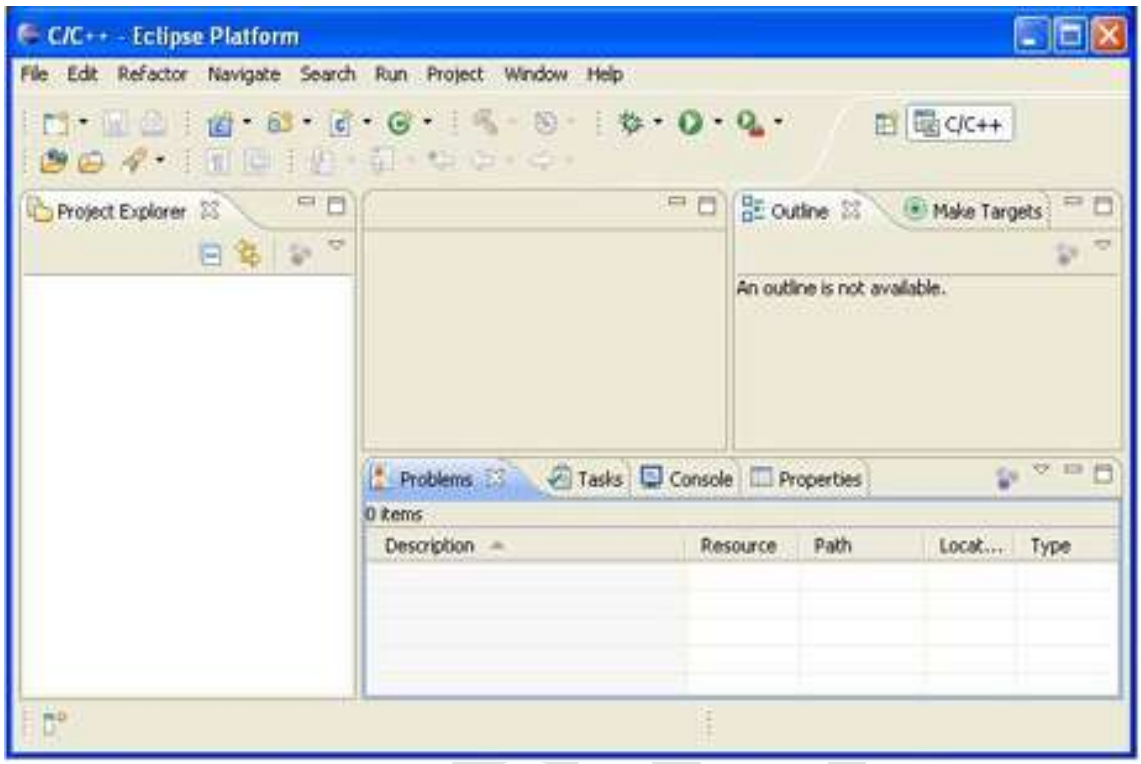
第一次启动会提示设置工作目录。 就是用户的工程存储的位置， 默认是

C:\Jennic\Application

建议不要修改这个目录

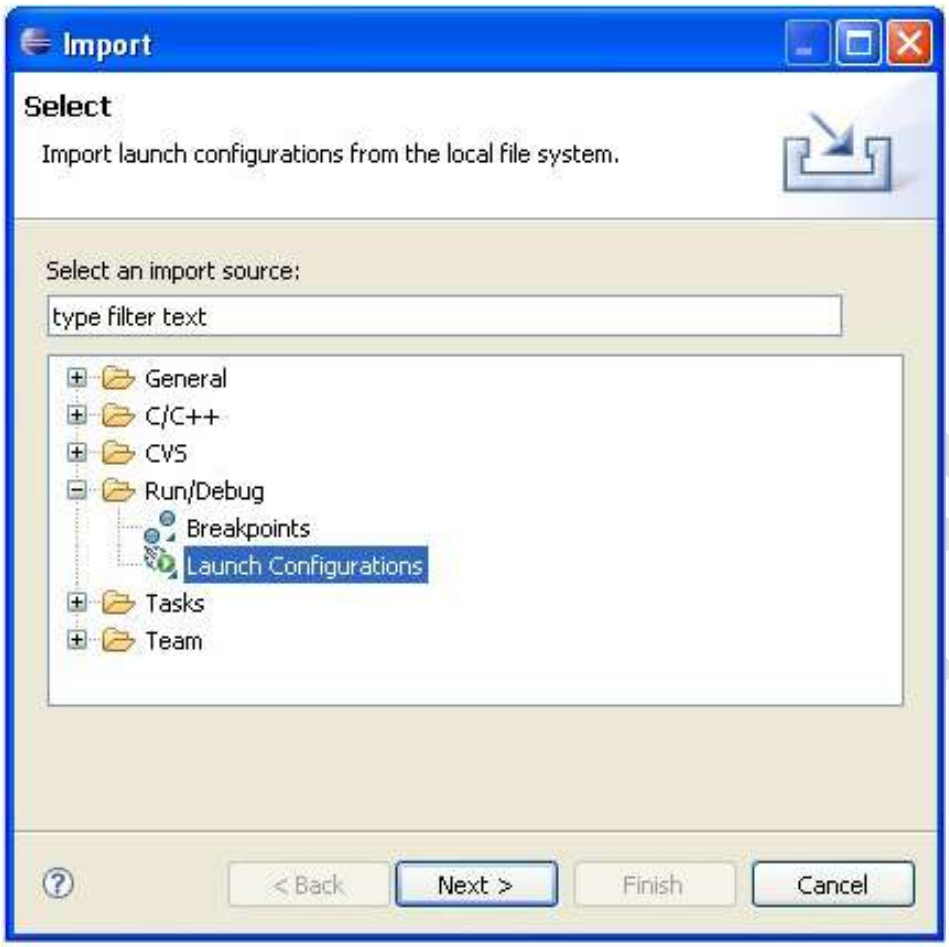
确认之后是欢迎界面

点右上角关闭按钮，关闭欢迎界面， 就进入了 Eclipse 的开发环境了

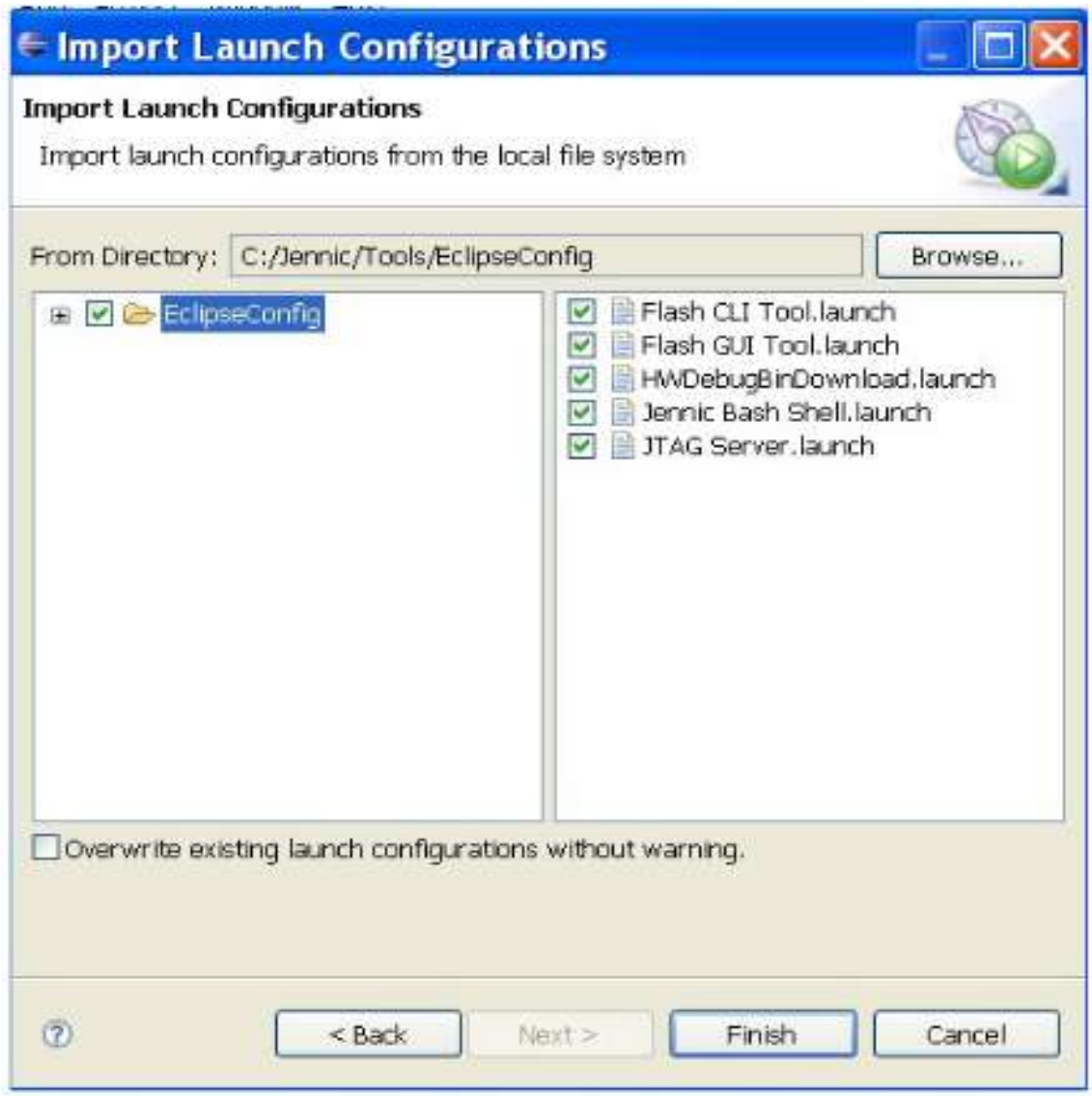


接下来需要安装 Jennic 的扩展工具。

选择 File->Import 打开导入对话框



选择 Run/Debug ->Launch Configurations 然后点击 Next>



点击 Browse…浏览到 C:\Jennic\Tools\eclipse_config 目录。 选中右边列表中的所有工具。
点击 Finish 按钮

这样就基本完成了开发环境的设置了。

下面介绍如何创建工程， 编译工程。

基本上我们不能直接从 Eclipse 创建 Jennic 的工程， 所以通常我们都需要用相应的基础项目模板来创建 Jennic 的工程。

在北京博讯科技有限公司网站有基于各个协议的项目模板， 建议用户将这些模板功能都下载下来备用。

http://www.boccn.com.cn/getfile.aspx?id=1036

这个是基于 802.15.4 的 JN516x 的项目模板- JN-AN-1174

http://www.boccn.com.cn/getfile.aspx?id=269

这个是基于 JenNet-IP 的项目模板- JN-AN-1190

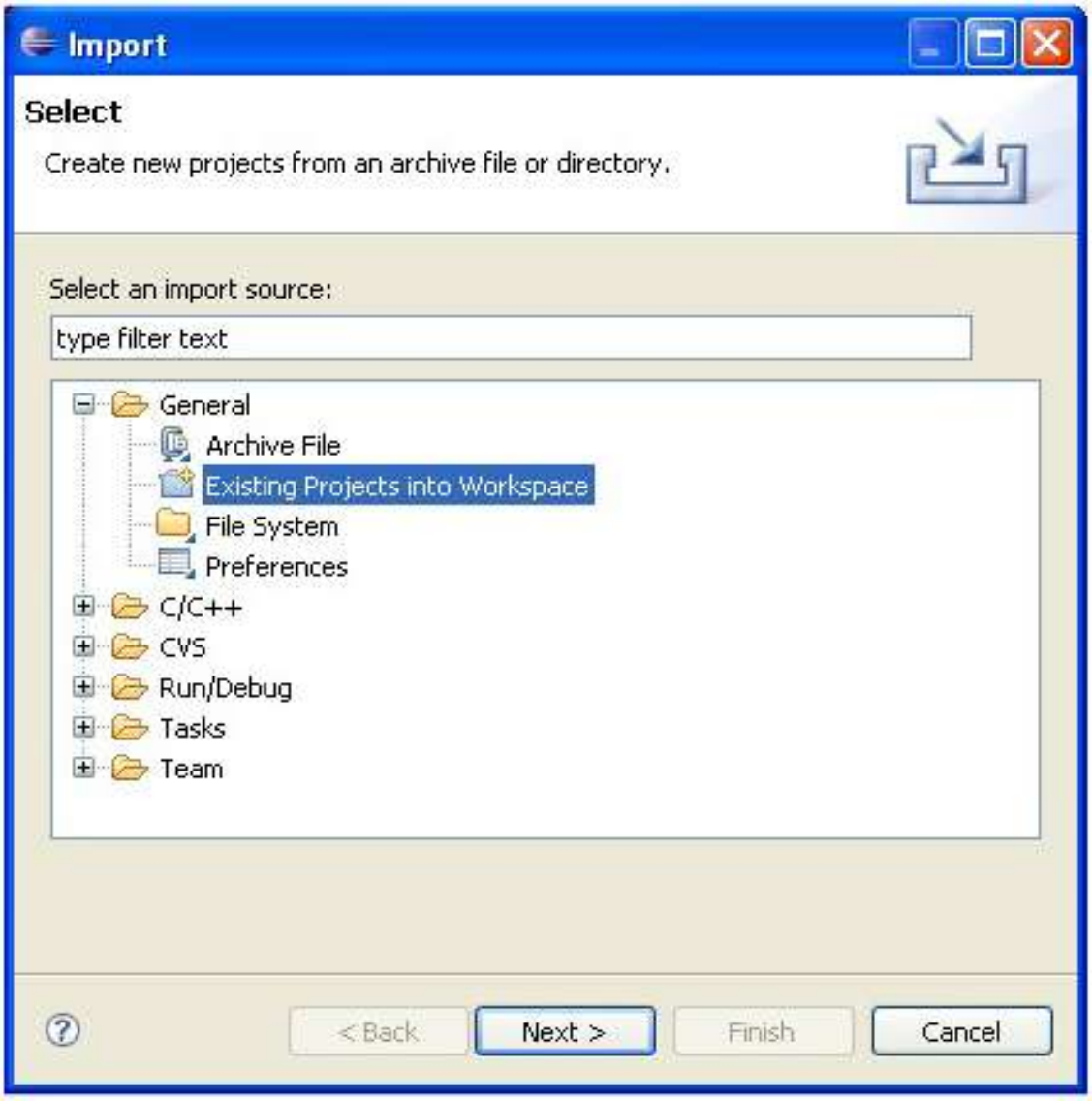
http://www.boccn.com.cn/getfile.aspx?id=1043

这个是基于 ZigBee-RF4CE 的项目模板- JN-AN-1200

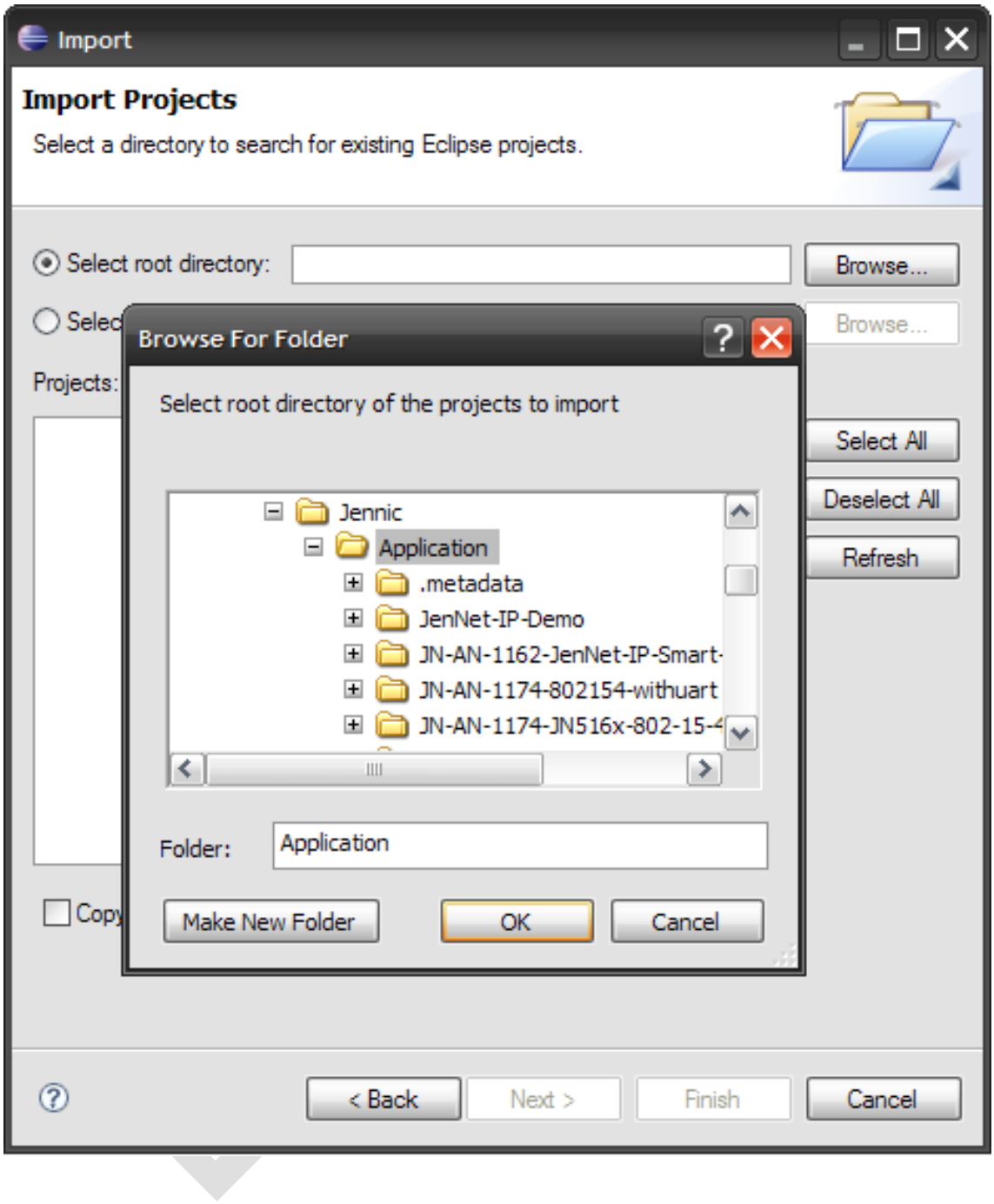
将这些模板下载之后， 当需要创建新的项目的时候就可以解压相应的 zip 文件， 然后修改解压后的目录名为自己的项目名称。 再把整个项目文件夹都移动到 C:\Jennic\Application 目录下面

然后打开 Eclipse， 选择 File->import

选择 General -> Existing Projects into Workspace 点击 Next>



Select root directory: 点击 Browse..按钮选择 C:\Jennic\Application 目录



确认之后， 就可以在 Projects:列表中看到还没有加入 workspace 的项目了。 选择相应的项目目录， 然后点击 Finish.

下面说说如何编译 Jennic 工程生成相应的 bin 文件。 Jennic 程序是以 bin 文件作为编译的结果的， bin 文件才可以通过 Flash Programmer 下载到测试板中运行。

每一个 Jennic 项目通常可以按照项目的配置编译出多个 bin 文件。 比如运行于 coordinator 或者 router 或者 end device 的 bin 文件。

下面我们以一个实际的例子来看怎么编译和下载 bin 文件。

请用户下载下面这个范例工程

http://www.boccn.com.cn/getfile.aspx?id=1037

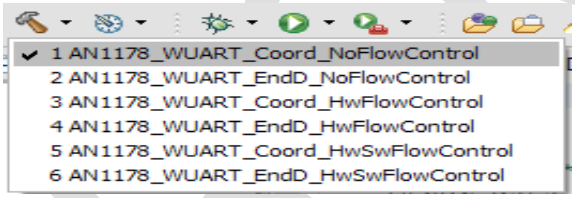
这是基于 802.15.4 的一个无线串口例程（JN-AN-1178）。 非常适合我们目前使用的开发包测试板进行演示测试。

请按照前面的导入工程的方法将这个项目下载， 解压， 并导入到 Eclipse 的 workspace 中。

导入之后， 选中左边 project explorer 相应的项目

JN-AN-1178-JN516x-Wireless-UART

然后点击工具栏上的小锤子图标， 可以看到下拉列表中的多个编译目标。



分别选中最上面的两个， 编译相应的 bin 文件。

这两个的编译结果是无线串口的非流控的 coordinator 和 enddevice 的两个 bin 文件。

在下面目录

C:\Jennic\Application\JN-AN-1178-JN516x-Wireless-UART\AN1178_154_WUART_Coord\Build

C:\Jennic\Application\JN-AN-1178-JN516x-Wireless-UART\AN1178_154_WUART_EndD\Build

找到 bin 文件

AN1178_154_WUART_Coord_JN5168_NO_FLOW_CTRL.bin

AN1178_154_WUART_EndD_JN5168_NO_FLOW_CTRL.bin

下面介绍如何用 Flash Programmer 来把 bin 文件下载到开发板中。

首先请准备两个开发板， 一个将作为 coordinator 一个将作为 enddevice。

用 USB 线将这两个开发板都链接到 PC 的 USB 口。

如果你之前没有安装相应的 USB 转串口驱动， 这个时候应该会提示安装驱动程序。

注意调整开发板的各个跳线， 以便程序可以正常的下载运行。

开发板左侧的串口切换跳线请确认是下面状态， 这样编程的 USB 口就是连通了 UART0

SW2 跳线请确认跳到 USB 供电模式。 这样就不需要安装电池。 如果需要电池供电， 可以讲这个跳线跳到 BAT 模式

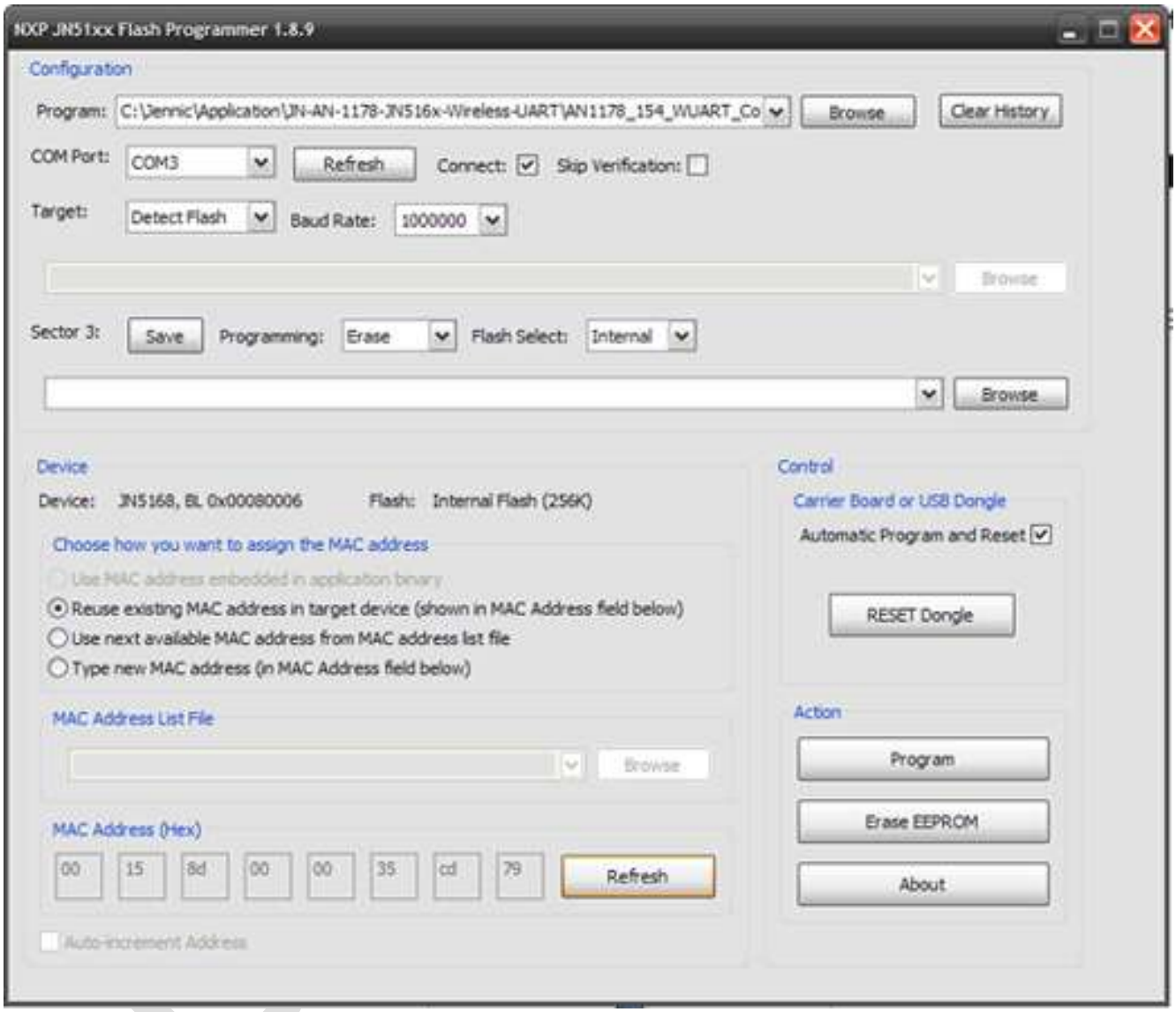
SW3 跳线请确认跳到 PGM 模式

运行所下载的独立的 Flash Programmer JN-SW-4007

目录中的 FlashGUI.exe

不要使用 ToolChain 中的 flash programmer, 因为这个版本比较老， 可能无法向最新的模块下载程序。

目前最新的版本是 1.8.9



点击 Browse 按钮选择相应的 bin 文件

COM Port 选择相应的 com 口

首先点击 Refresh 读取一下 MAC 地址， 看看是否可以和相应的板卡通讯。

然后点击 Program 下载程序。

稍等片刻， 程序就下载到板卡里面了

这里有一个小提示

可以将 Flash Programmer 的 Baud Rate 设置的大一些， 这样可以提高下载速度， 加快调试的进度。

按照上面的步骤， 选择另一个文件和 com 口 将另一块测试板也下载好。

关闭 flash programmer. 因为我们需要测试的是无线串口， 所以需要使用串口， 如果不关闭 flash programmer 的话会出现串口被占用的情况。

打开两个超级终端， 分别设置相应的串口， 和 115200-8-n-1, 无流控。

在两个超级终端输入数据， 看另一个终端的显示。 这一范例演示了基本的无线串口透传的应用。

至此， 我们已经完整的介绍了从开发环境的安装， 项目的导入建立， 编译， 下载过程了。

从下一章开始我们将具体的介绍相应的协议栈， 进行程序的开发说明。

因为目前 JN5168 支持 802.15.4， JenNet-IP， ZigBee SE, ZigBee HA, ZigBee LL, ZigBee RF4CE 多种协议。 用户可以根据自己的项目需要选择合适的协议平台进行应用的开发。

各种协议的对比可以参考下面表格

	IEEE 802.15.4	JenNet-IP	ZigBee Pro
网络拓扑类型	星型， 点对点	树状， 链状， 星型	Mesh 网络
支持最大节点数量	50	500	50
网络自恢复	不支持	支持	支持
用户代码可用空间	Coordinator: 115KB EndDevice: 115KB	Coordinator: 85KB Router: 85KB EndDevice: 95KB	Coordinator：36KB Router: 36KB EndDevice: 48KB
协议栈标准	IEEE 802.15.4 标准	基于 IEEE 802.15.4 的 JenNet-IP 协议 IETF IP、 UDP、 6LoWPAN	基于 IEEE 802.15.4 的 ZigBee Pro 协议
ZigBee License	不需要	不需要	需要
应用领域	线缆取代 远程控制 物流管理 语音通话	远程控制 智能灯控 安全 健康医疗 楼与自动化 环境监控 智能能源管理	智能能源管理 家庭自动化 智能灯控

所以后面的基于协议开发的章节， 用户可以按照需要跳着查阅， 不需要按照章节的顺序阅读。

3：基于 802.15.4 协议栈进行开发

首先需要说明的是， JN516x 系列芯片支持基于多种协议进行开发。 802.15.4， JenNet-IP, 和 ZigBee 的一系列协议。 用户可以根据自己的应用需要选用协议平台。 不管您采用什么协议进行开发， 都是使用 C 语言， 开发环境和面对的硬件接口也都是一致的。 因为硬件接口的一致性， 所以如果你一开始选用了一种协议进行了开发， 今后也可以向其他协议进行应用的移植。

那么， Jennic 的 802.15.4 协议有什么特点， 什么情况需要采用这种协议进行开发呢？

Jennic 的 802.15.4 协议是一个基础的网络协议， 协议本身实现了基本的星形网络拓扑结构，协议本身实现了基本的无线网络创建和节点的管理。 协议本身不具备复杂的网络拓扑结构，所以如果用户需要实现树状或者网状网络拓扑就需要自己编写代码管理网络结构。开发的灵活性也足够大，而且从所开发出的最终产品的成本上来说，可以为每个网络节点省出一点 ZigBee 协议栈的版权使用费用。

相对来说， Jennic 的 802.15.4 协议的 API 结构比较简单， 网络节点类型也只有 coordinator 和 enddevice 两种， 软件的开发入门比较容易， 编译出来的应用也相对较小。 所以如果您的应用如果不需要复杂的网络拓扑， 或者希望开发自己的网络拓扑管理， 那么基于 802.15.4 的协议栈 API 开始开发是比较合适的。

本章将以 JN-AN-1174 IEEE802.15.4 应用程序模板为例子， 讲解基于 802.15.4 协议进行开发的要点。

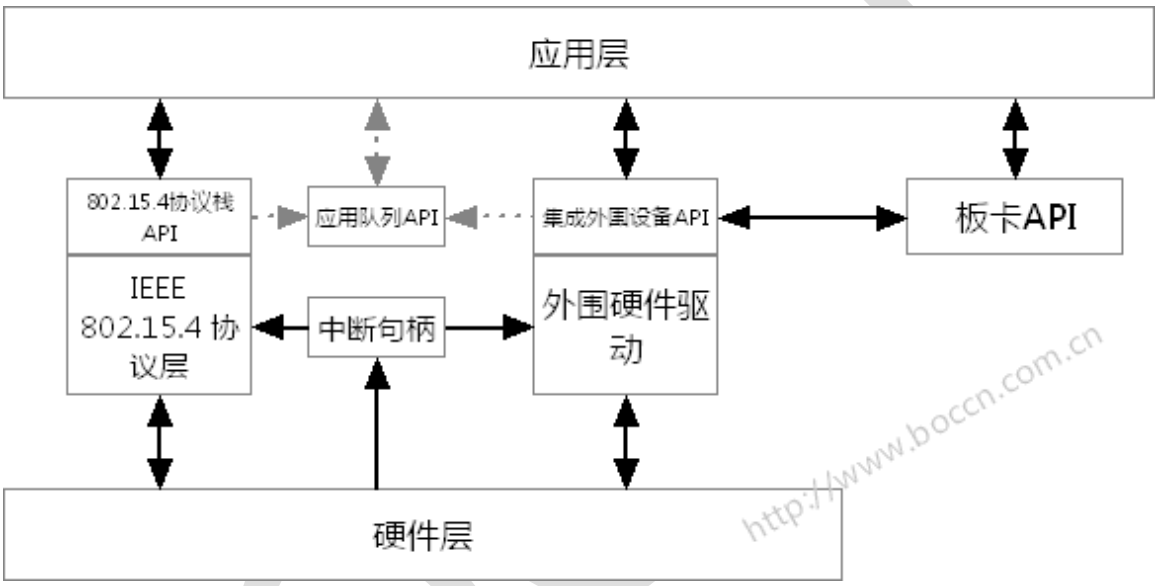
开始之前， 您最好下载这个应用模板， 并按照上一章讲解的方法将这个范例解压并导入 Eclipse 开发环境。

http://www.boccn.com.cn/getfile.aspx?id=1036

3.1 IEEE 802.15.4 协议栈的架构，接口和中断说明

IEEE 802.15.4 的协议栈架构概述，开发人员可以参考下图理解 802.15.4 的协议栈架构

802.15.4 的协议栈架构



- 应用程序通过 802.15.4 的协议栈 API 和 IEEE 802.15.4 的协议层进行交互。这一交互用来实现 MCPS/MLME 的请求和确认，消息的标识和回应。IEEE 802.15.4 协议层和更底层的硬件以及寄存器进行交互。
- 应用程序通过集成外围设备 API 和芯片上集成的外围设备（比如 AD，DA，DIO,Timers...）进行交互.这一 API 访问硬件寄存器
- 应用程序通过板卡 API 和开发包中的传感器板或者控制器板访问板卡上的设备，比如传感器等等。
- 硬件层产生各种中断通过中断句柄将其转发给各个软件模块
- 开发人员还可以使用我们提供的应用队列 API 来简化对于通讯协议层和硬件 API 中断的处理过程。（可选用，JN-AN-1174　IEEE802.15.4 应用程序模板使用了这一队列接口，建议用户的应用也使用）

802.15.4 协议栈 API

我们提供的 802.15.4 的协议栈 API 可以使应用访问 802.15.4 的协议栈以及控制基于 JN516x 无线微控制器的 IEEE 802.15.4 MAC 层。详细的 API 说明可以参考 JN-UG-3024

集成外围设备 API

集成外围设备 API 可以使应用程序创建控制以及处理 JN516x 无线微处理器的集成外围设备。比如 UART，计时器和通用 IO

详细的 API 说明可以参考 JN-UG-3087

板卡 API

板卡 API 可以使应用程序使用 无线传感器板和无线控制器板的板载设备，比如 LCD 屏，LED，按钮，温湿度传感器等等。这一 API 封装了对于硬件寄存器和中断的处理

详细的 API 说明可以参考 JN-RM-2003

应用队列 API

开发人员在开发无线应用的时候可以选择性的使用我们提供的 应用队列 API(Application Queue API).这一系列的 API 负责处理各种中断，并向应用层提供了一个基于队列的接口，这样开发人员就可以不用自己处理各种中断的回调函数。当来自下层的中断产生的时候，中断的入口就会被分类压入到三个专用的队列中（MLME（网络管理队列），MCPS(网络数据队列），硬件事件队列））应用程序灵活的掌握时机从队列中取出事件进行处理。

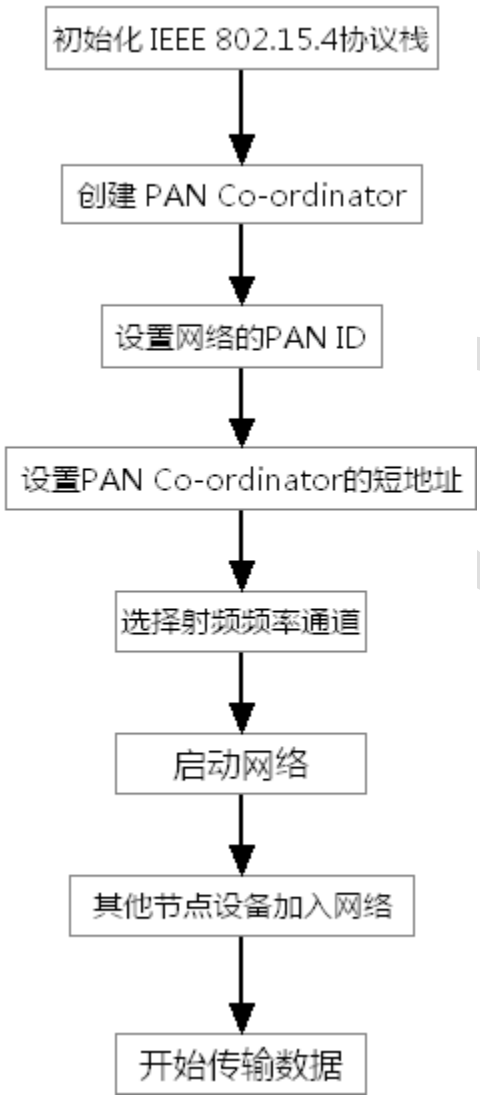
应用队列 API 也容许开发人员使用传统的方法注册自己的回调函数，但是我们强烈推荐开发人员采用应用 API 来处理队列事件以加快开发速度并同时保证程序的稳定性。

3.2 IEEE 802.15.4 网络的建立过程

这篇文章将详细的描述 IEEE 802.15.4 无线网络的建立过程，我们描述的是一个星形网络的建立过程。需要注意的是这里我们所建立的网络是非信标网络。

概述

下图描述了一个 IEEE 802.15.4 网络的建立过程



下面将详细解释一下整个网络的建立过程。

建立网络的过程

1.首先，每一个设备的 IEEE 802.15.4 的协议栈必须要对其 PHY 和 MAC 层进行初始化的工作。

2.创建 PAN Co-ordinator，每一个网络必须有一个也只能有一个 PAN Co-ordinator,建立网络的第一个步骤就是需要选择并且初始化这个 Co-ordinator.初始化 PAN Co-ordinator 的动作只在相应的被事先约定的设备上进行。

3.选择 PAN ID 和 Co-ordinator 的短地址

PAN Co-ordinator 一旦初始化完成就必须为它的网络选定一个 PAN ID 作为网络的标识。PAN ID 可以被人为的预定义。

这里需要说明的是 PAN ID 可以通过侦听其他网络的 ID 然后选择一个不会冲突的 ID 的方式来获取。PAN Co-ordinator 可以扫描多个频率通道，当然开发人员也可以指定设备优先扫描指定的通道来确定不和其他网络冲突的 PAN ID。

每一个 PAN Co-ordinator 设备都已经具有了一个唯一的固定的 64 位 IEEE MAC 地址，通常我们叫做扩展地址。但是作为组网的标识他还必须分配给自己一个 16 位的网络地址，通常我们叫做短地址。使用短地址进行通讯可以使网络通讯更轻量级更加高效。这一短地址是由开发人员预先定义的，PAN Co-ordinator 的短地址通常被定义为 0x0000.

4.选择射频频率

PAN Co-ordinator 必须选择一个网络所建立的射频频率通道。PAN Co-ordinator 可以通过进行一次能量扫描检测来找到一个相对安静的通道。通过通道能量扫描检测 API 将返回每一个通道的能量水平，能量水平高就标志着这个通道的无线信号比较活跃。接下来 PAN Co-ordinator 就可以根据这些信息选择一个可以利用的通道来建立自己的无线网络。

5.启动网络

一个无线网络的启动过程是从初始化配置 PAN Co-ordinator 开始的，然后这个设备就将以 Co-ordinator 的模式启动。然后 PAN Co-ordinator 就将开放对于加入网络的请求应答。

6.设备加入网络

一旦网络中出现了可以利用的 Co-ordinator，其他的网络设备就可以加入网络了。一个设备如果需要加入网络首先其要完成自己的初始化过程，然后他需要找到 PAN Co-ordinator.

为了找到 PAN Co-ordinator，设备需要进行频道扫描，它将在特定的频率通道中发送信标请求。当 PAN Co-ordinator 检测到信标请求后，Co-ordiantor 将回应相应的信标来向设备标识自己。

对于信标网络（所谓的信标网络就是 PAN Co-oridnator 将周期性的发送信标),需要加入网络的设备可以被动的侦听来自 Co-oridnator 的信标。

设备找到 PAN Co-ordinator 之后就将发出加入网络的申请.Co-ordinator 将决定是否具有足够的资源接受新的设备，并且决定是否接受和拒绝设备加入网络。

如果 PAN Co-ordinator 接受了设备，它将发送一个 16 位的短地址给设备，作为设备在网络中的标识。

在设备之间传输数据

当网络中出现了 PAN Co-ordinator 和至少一个端节点设备后，网络就可以进行数据传输了。数据传输的过程如下所述。

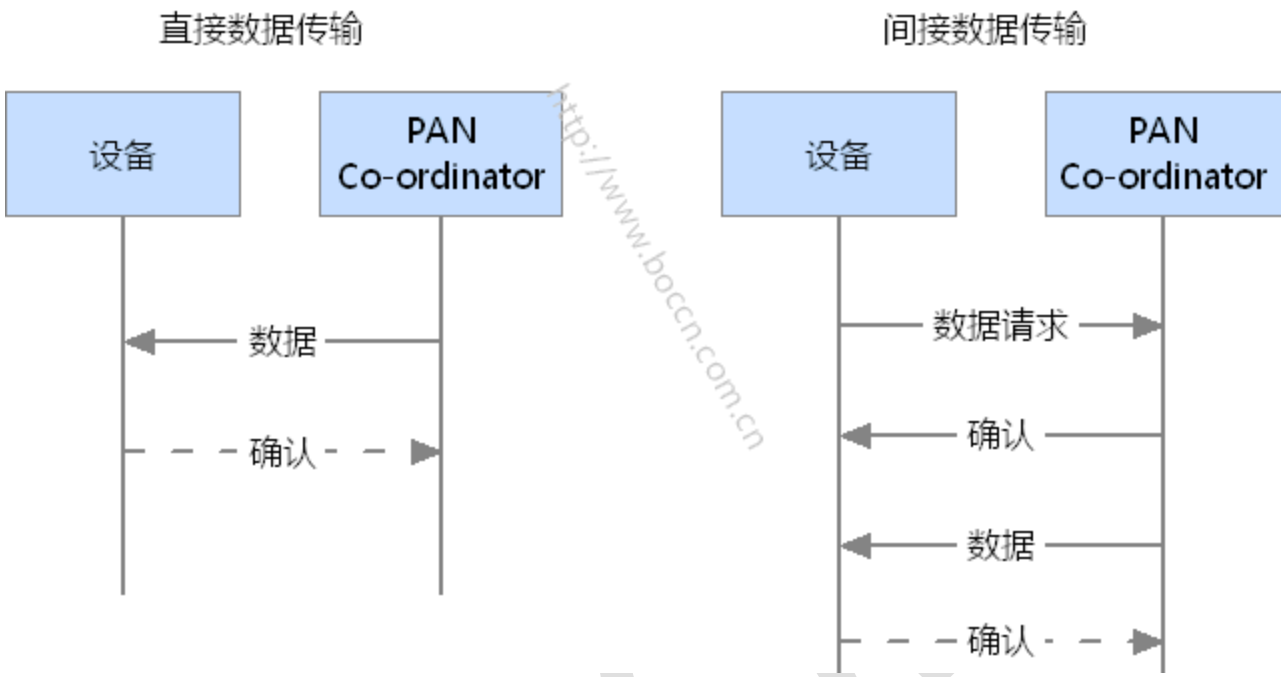
Co-ordinator 向 End Device 传输数据

有两种方法可以实现 Co-ordinator 向 End Device 传输数据：

1.直接传输：PAN Co-ordinatro 可以将数据直接发送给 End Device。End Device 接收到数据后可以发送确认消息给 Co-ordinator.这种数据传输方式就要求 End Device 随时都处于数据接收的状态，也就是要求其随时都要处于唤醒的状态。后面我们将详细的解释这种工作方式。

2.间接传输: 另外一种传输方式就是 Co-ordinator 可以将数据保存起来等待 End Device 请求读取数据。采用这种方式，End Device 为了获得数据必须先要发送数据请求。发送数据请求后，Co-ordinator 就会判断是否有需要发送给这个设备的数据，如果有就发送相应的数据给 End Device。接到数据的设备将发送确认信息。 这一方式适用于 End Device 设备需要较低功耗的情况，其大部分的工作状态都处于休眠状态以节省能量。

以上所述的数据传输方式可以用如下图表示：



End Device 向 Co-ordinator 传输数据

End Device 通常向 PAN Co-ordinator 直接发送数据，Co-ordinator 接到数据后可以发送确认消息。

3.3 应用程序的代码框架

我们所讲解的范例代码需要以下的环境：

- 您应该有一个开发板作为 PAN Co-ordinator
- 您应该至少再有一个开发板作为 End Device
- 您可以使用预先定义的 PAN ID 和短地址作为范例程序的网络参数，作为一个演示的应用这些参数已经可以演示基本的概念了，但是如果您需要开发一款实际的产品或者项目，您应该规定自己的网络地址的分配逻辑。
- 我们将采用星形的网络拓扑结构
- 我们将采用非信标的网络(这就意味着 PAN Co-ordinator 将不会周期的发送信标)
- 我们将使用短地址来标识网络中的设备
- 数据的传输模式采用直接传输，并且进行数据的接收确认
- 我们将不采用任何的安全措施

本节详细的讲解应用模版代码，我们将先介绍支持文件，然后再介绍代码中的函数调用。

如果你的开发环境还没有安装好建议您先按照第二章的内容在电脑上部署您的开发环境。

并且导入 JN-AN-1174 的范例工程。

在 Eclipse 中展开这个工程的目录， 我们将对照实际的目录进行文件结构的说明。

文件的说明：

在这个范例程序有四个目录：

- Common/Source/config.h: 这个头文件包含了一些网络参数设置，比如 PAN ID，短地址，和需要扫描的通道。
- AN1174_154_Coord 目录是 Coordinator 的相应文件， Source 是源代码， Build 是编译后的 bin 文件存放处。Make 文件也放在 Build 目录
- AN1174_154_EndD 目录是 EndDevice 的相应文件， Source 是源代码， Build 是编译后的 bin 文件存放处。Make 文件也放在 Build 目录。
- Doc 目录有这个范例的说明文档

代码描述

这部分内容从函数的级别详细解释了代码。我们将分别解释 PAN Co-ordinator 和 End Device 的代码。

config.h 头文件将被引用到两个源代码文件中，同时两个源代码文件也引用了以下的头文件：

jendefs.h, AppHardwareApi.h, AppQueueApi.h, mac_sap.h, mac_pib.h

AN1174_154_Coord.c 的内容

开发者最常问的问题之一就是为什么 Jennic 的程序都没有 Main 函数，这个熟悉的函数哪里去了呢？这是因为 Jennic 程序都由 boot loader 来启动和引导，boot loader 引导完成后就将自动的调用 AppColdStart 函数，您可以认为 AppColdStart 就是我们通常所说的 Main().

AppColdStart 将进行下面的操作：

1.AppColdStart 将调用函数 vInitSystem(),这一函数将完成以下任务:

- 初始化设备的 IEEE 802.15.4 的协议栈
- 设置 PAN ID 和 PAN Co-ordinator 的短地址，在这个应用中这些参数都由我们预定义在 config.h 这个文件中
- 打开射频接收器
- 使 Co-ordinator 可以接受其他的设备加入网络

2.AppColdStart()会调用 vStartEnergyScan(),这一函数将会开始在各个通道进行能量扫描以获得各个通道的能量级别.所扫描的通道以及速率都定义在 config.h 中。扫描将通过初始化一个 MLME 请求并将其发送给 IEEE 802.15.4 的 MAC 层来实现。

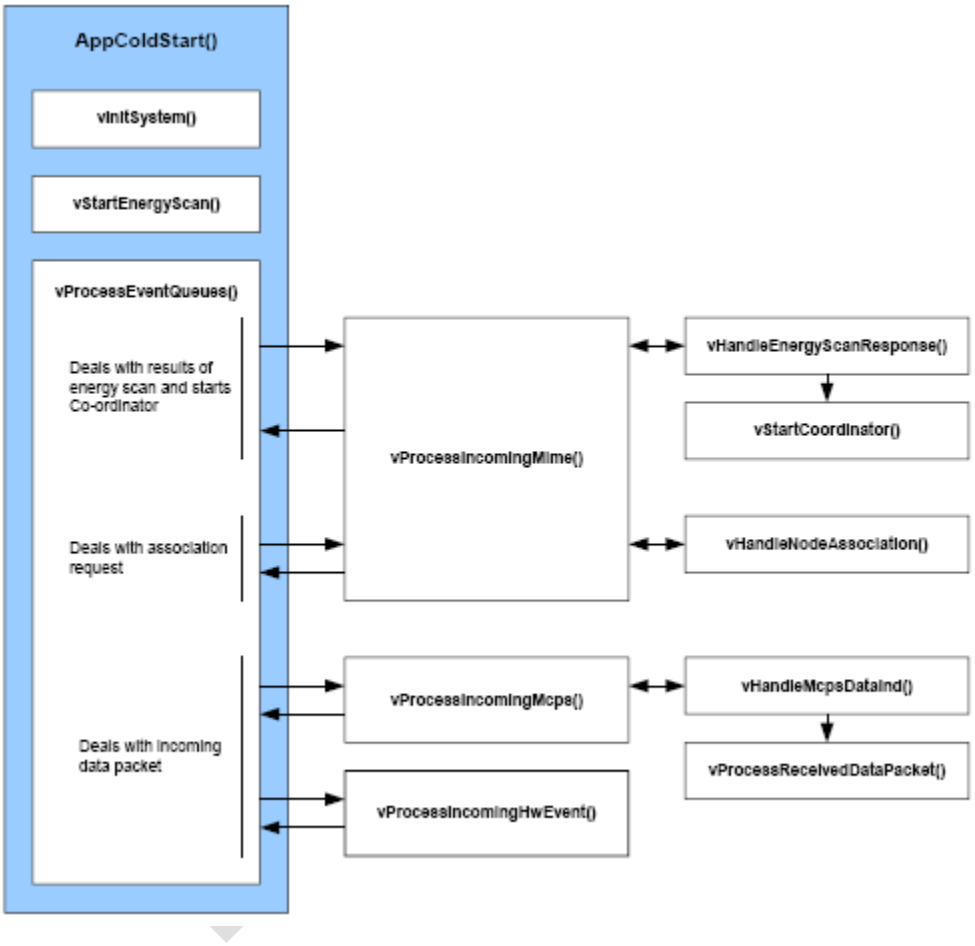
3.AppColdStart()将通过调用 vProcessEventQueues()的方式等待 MLME 的回应。vProcessEventQueues()函数将检查三个不同类型的事件队列并将接到的事件交给不同的事件处理函数处理。比如这个函数将调用 vProcessIncomingMlme()函数来处理 MLME 回应. 而这个函数将调用 vHandleEnergyScanResponse()来处理能量检测扫描的回应结果。这个函数将检查所有通道的能量级别，并挑选一个最安静的通道作为建立网络的通道。接下来将调用 vStartCoordinator()函数，这个函数将设置必要的参数并且递交 MLME 请求来启动网络，启动网络的请求不需要处理任何的回复信息。

4.AppColdStart()循环调用 vProcessEventQueues()来等待其他设备的加入网络的请求，入网请求将以 MLME 请求的方式发送到 coordinator.当请求到达的时候函数将调用 vHandleNodeAssociation 来处理。接下来 coordinator 将创建并发送入网请求回复。

5.AppColdStart 将循环调用 vProcessEventQueues 来处理来自于 MCPS 的消息队列和来自于硬件的消息队列.

- 当数据到达 MCPS 队列后，vProcessEventQueues 首先调用函数 vProcessIncomingMcps()来接收到达的数据帧.vProcessIncomingMcps()调用 vHandleMcpsDataInd()，这个函数将调用 vProcessReceivedDataPacket，在这个函数里面您可以自定义您自己的数据处理过程。
- 当硬件事件到达硬件队列后，vProcessEventQueues 将调用函数 vProcessIncomingHwEvent 来接收到来的事件。您需要在这个函数中自定义自己的事件处理过程。

您可以参考下面的示意图来理解



AN1174_154.EndD.c 的内容介绍

End Device 的运行过程仍然是从 AppColdStart 开始。这一函数和 Co-ordinator 的运行方式完全的不同，下面将详细的讲解这个过程。

1.AppColdStart 调用 vInitSystem，这个函数将初始化 IEEE 802.15.4 的协议栈

2.AppColdStart()调用 vStartActiveScan()开始对于活动通道的扫描, End Device 将向扫描的通道发送信标请求,并接收 PAN Co-ordinator 的信标请求回应。需要扫描的通道和速率将在 config.h 中定义。扫描请求的初始化和发送的工作可以通过 MLME 请求的方式通过 IEEE 802.15.4 的 MAC 层发送。

3.AppColdStart()将通过 vProcessEventQueues 来检查和处理 MLME 回应。这个函数将调用 vProcessIncomingMlme()来处理收到的 MLME 回应。vHandleActiveScanResponse()会被调用处理返回的活动通道扫描结果：

- 如果找到 PAN Co-ordinator，函数将保存相应的 Co-ordinator 信息(比如 PAN ID,短地址,逻辑通道),并且调用 vStartAssociate()向 Co-ordinator 来提交入网请求，这一请求将通过 MLME 请求的方式提交.
- 如果 PAN Co-ordinator 没有被找到(可能是由于 Co-ordinator 还没有初始化完成）.这一函数将重新调用 vStartActiveScan()来重新启动扫描。

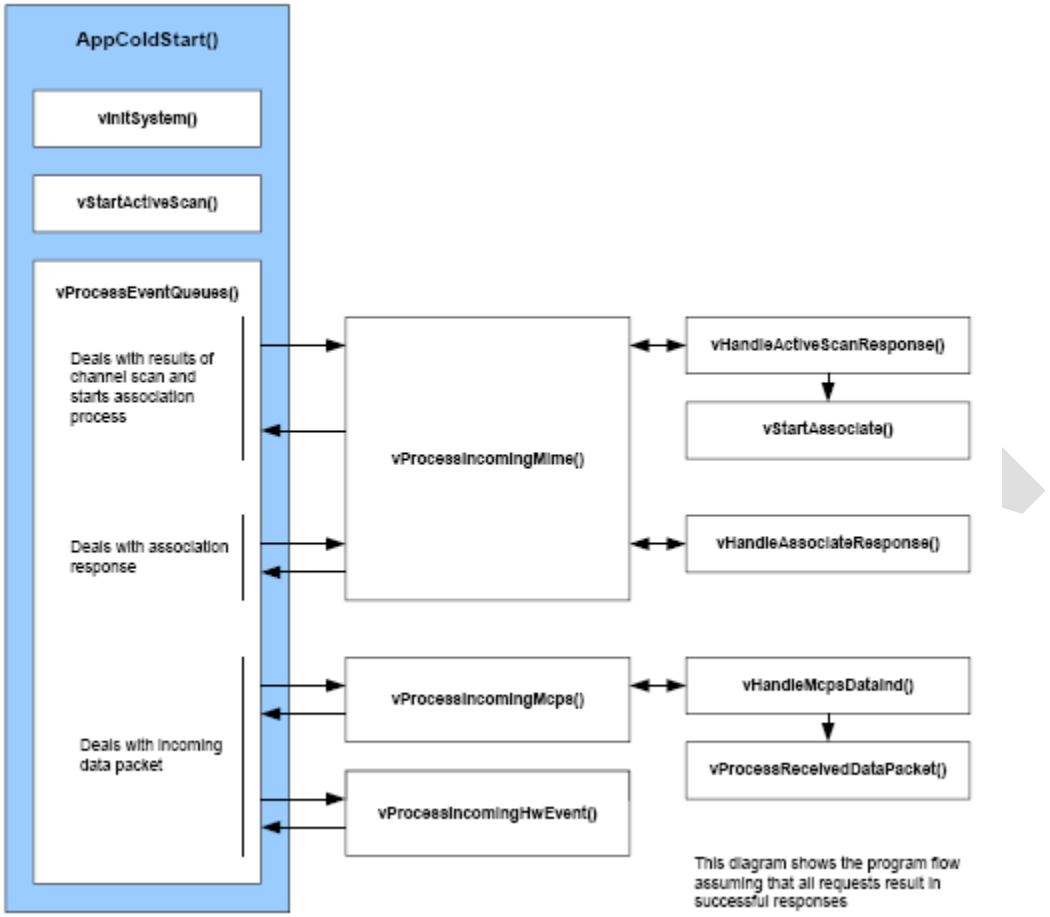
4. AppColdStart 将循环的调用 vProcessEventQueues()等待来自 Co-ordinator 的入网回复。当收到回复后就将调用 vProcessIncomingMlme(),然后将调用 vHandleAssociateResponse 来处理回复，接下来的函数将检查回复的状态：

- 如果 PAN Co-ordinator 接受的入网请求，将设备置于联网状态。
- 如果 PAN Co-ordinator 拒绝了入网的请求，函数就将重新调用 vStartActiveScan()来开始搜索另外一个 PAN Co-ordinator。

5. AppColdStart()接下来将循环的调用 vProcessEventQueues 来等待来自于 PAN Co-ordinator 的 MCPS 信息或者硬件的队列信息。

- 当数据到达了 MCPS 队列，vProcessEventQueue()首先使用函数 vProcessIncomingMcps()来接收数据帧，接着调用 vHandleMcpsDataInd()，接着调用 vProcessReceivedDataPacket()，开发人员可以在这个函数里面编写自己的数据处理过程。
- 当硬件事件到达硬件事件队列,vProcessEventQueues()将调用 vProcessIncomingHwEvent()来接收到达的事件，您可以在这个过程中编写自己的事件处理逻辑。

下面的图表示了 End Device 的工作过程。



如何调整骨架代码：

下面的内容提供了一些指导来告诉开发人员如何利用骨架代码进行修改以符合不同的应用需求。内容包括：

- 如何修改预定义的 PAN ID？
- 如何修改预定义的短地址？
- 如何向网络中添加新的 End Device？
- 如何修改通道扫描？
- 如何定义数据包的接收过程？
- 如何编写数据传输过程？
- 如何向项目加入其他 C 文件？

如何修改预定义的 PAN ID？

在我们提供的范例骨架代码中 PAN ID 被定义在 config.h 中，他被预先定义为 0xCAFE.

如果您需要使用不同的 PAN ID,打开 config.h 您就可以将 PAN ID 修改为您需要的 16 进制地址

```
#define PAN_ID              0xCAFE
```

需要注意的是所定义的 PAN ID 不要和在同一区域内的其他基于 IEEE 802.15.4 的网络冲突。

如何修改预定义的短地址

对于 PAN Co-ordinator 和 End Device 的短地址定义也都在 config.h 中。在我们提供的范例代码中，Co-ordinator 的短地址被定义为 0x0000 而第一个 End Device 的地址被定义为 0x0001. 后面这个地址也是 End Device 的起始短地址,如果你需要增加多个 End Device，那么短地址将自动的被生成分配，每一个短地址都比上一个增加 0x0001.

如果您需要使用不同的短地址，打开 config.h 并且修改下面的定义

```
#define COORDINATOR_ADR                         0x0000
#define END_DEVICE_START_ADR            0x0001
```

需要注意的是，我们通常都是设置 PAN Co-ordinator 的地址为 0x0000,您在应用中最好也遵从这个约定。

如何让更多的 End Device 加入网络

我们提供的范例代码被设计为网络中最少有两个节点；一个 PAN Co-ordinator 和 End Device.默认的这个范例代码最大可以接受 10 个 End Device，这就是说您可以使用 10 个 End Device 加入网络而不用修改任何代码。 如果您需要使用更多的 End Device，您可以参考下面的说明来修改代码。

修改 config.h 文件
首先您需要修改 config.h 文件来定义所能接受的最大节点数量的数值

```
#define MAX_END_DEVICES                 10
```

修改 AN1174_154_EndD.c 文件

这个文件是 End Device 的源代码文件，如果您所使用的新的 End Devie 具有一些特殊的功能和特性，比如额外的温湿度传感器，或者 AD，DA 的功能，那么您需要修改这个文件来生成新的设备。

关于 AN1174_154_Coord.c 文件

这一文件是 Co-ordinator 的源代码文件，一般来说您不需要修改这个文件来接收新的 End Device 节点的加入。

如何修改通道扫描

我们提供的范例代码提供了两种通道扫描的模式

- 能量检测扫描，PAN Co-ordinator 在建立网络的时候会进行这个扫描来选择合适的通道建立网络
- 活动通道扫描，End Device 在需要加入网络的时候进行这个扫描来找到通道中的 PAN Co-ordinator

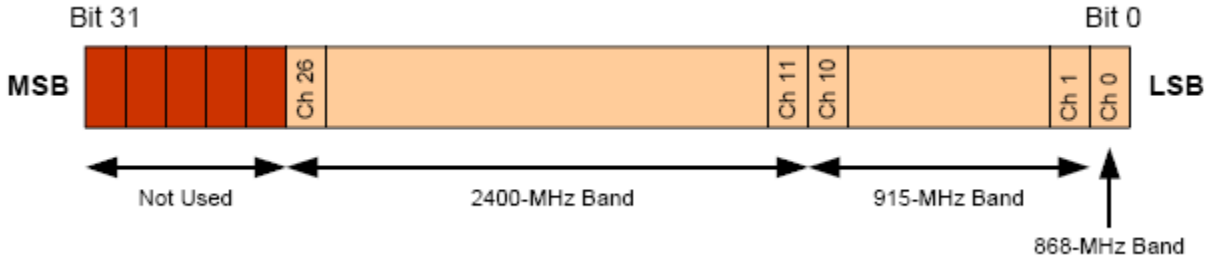
通常情况下是没有必要检查所有可能的频率通道的。IEEE 802.15.4 所定义的 27 个通道分布在三个频段(868,915 和 2400MHz). 因为网络只能建立在某一个频段内，所以扫描其他的两个频段的通道就没有太大的意义。对于 Jennic 产品而言，通常工作在 2400 MHZ 频段，也就是 11 到 26 通道。而且通常来说您在开发的过程中也可以发现一个特定的通道总是被本地的一些无线网络所占用，所以对于这些通道的扫描也可以从通道扫描中排除。您可以预先定义需要扫描的通道，而且可以定义每一个通道所检查的时间。这些参数都可以通过下面的定义实现。

定义所扫描的通道

找到 config.h 中下面的位置

```
#define SCAN_CHANNELS                    0x07fff800UL
```

SCAN_CHANNELS 参数定义了需要扫描的通道。每一个位都代表一个通道，最低的位代表通道 0。如果一个通道需要扫描那么相应的位就是 1，如果一个通道不需要扫描相应的位就是 0。 您可以参考下面这个图来理解参数的定义。



需要注意：SCAN_CHANNELS 的定义既被用于能量检测扫描也被用于活动通道扫描，另外 Jennic 的产品只能工作在 2.4G 这个频段，所以对于其他通道的扫描是没有意义的。

定义通道扫描速率

在 config.h 中找到下面的定义

```
#define ACTIVE_SCAN_DURATION                    3
#define ENERGY_SCAN_DURATION                    3
```

上面两个参数定义了相应的扫描时间，

ACTIVE_SCAN_DURATION 定义了 活动通道扫描的时间，ENERGY_SCAN_DURATION 定义了能量检测扫描的时间。这些参数定义了花在每一个通道的扫描时间，如果需要换算成 ms,可以参考下面的公式

Active Channel Scan
Channel scan duration(ms) =15.36 x ((2 ^ ACTIVE_SCAN_DURATION) + 1)

Energy Detection Scan

Channel scan duration(ms) =15.36 x ((2 ^ ENERGY_SCAN_DURATION) + 1)

打个比方说，如果值设置为 3，那么通道扫描的时间就使 138.24ms

需要注意的是，这两个参数必须被设置为 0 到 14 的整数，也就是说通道扫描的时间将在 30.72ms 和 251.6736s 之间

如何定义处理接收到的数据的过程？

IEEE 802.15.4 的协议栈会将接收到的数据压入到 MCPS 队列中。我们提供的范例代码不论是 PAN Co-ordinator 还是 End Device 都将从队列中取出数据，但是不对数据做任何的处理，您必须自己定义自己的数据处理过程。但是范例代码中提供了一个空的函数 vProcessReceivedDataPacket()来给你使用。您可以修改 AN1174_154_Coord.c 或者 AN1174_154_EndD.c 中这个函数的过程来定义自己的数据处理过程。

如何编写数据的发送过程？

在 AN1174_154_Coord.c 和 AN1174_154_EndD.c 这两个源文件中，都已经定义好了一个数据传输函数 vTransmitDataPacket().如果您需要传输数据只需要简单的调用这个函数就可以了。

如何向项目加入其他 C 文件

仅仅通过 Eclipse 向项目加入 C 文件是不能被自动加入编译的。

所以需要手动的修改相应的 make 文件

比如你需要编写自己的一些功能函数， 把 C 文件加入了 common/source 之后， 如果在 Coord 或者 EndDveice 的代码里面使用了相应的函数， 编译之前需要修改 make 文件

如果需要修改 coord 的 make 文件

在 AN1174_154_Coord/Build/makefile 中找到下面的段落

```
############################################################################
#######
# Application Source files

# Note: Path to source file is found using vpath below, so only .c
filename is required
APPSRC += AN1174_154_Coord.c
APPSRC += AppQueueApi.c

############################################################################
```

在这后面加上你新加入的 C 源文件。比如： `APPSRC += newfile.c`

这样才可以正常的编译， 如果需要修改 EndDevice 的代码也是同样的操作。

4：基于 Jennet-IP WPAN 协议栈进行开发

4.1 JenNet-IP WPAN 协议栈简介及基本概念

JenNet-IP 是 NXP 基于 802.15.4 协议进一步开发的一个结合无线网络和 IP 网络的网络协议。 主要面向 工业，商业和住宅领域， 提供物联网解决方案。

JenNet-IP 既可以实现独立的 WPAN 网络， 也可以实现 WPAN 和 IP 网络的互联互通。 协议提供了自组网， 高可靠性和可扩展的网络架构， 可支持高达 500 个无线网络节点。

为了便于后面学习的方便， 我们先介绍一些 JenNet-IP 网络基本概念

节点类型,

JenNet-IP 网络中有三种节点类型， 每种节点类型对应了节点在协议层的网络角色。 分别是

Coordinator: 网络的创建者， 一个无线网络中必须有也只能有一个 Coordinator 节点。 通常在网络的创建时定义网络的各种参数， 启动网络，并接受其他节点加入网络，管理网络其他节点。

Router: 路由节点， 可以加入 Coordinator 或者其他 Router 节点，也可以接受其他 Router 或者 End Device 节点的加入， 是数据的中转节点。 一般来说不能睡眠。

End Device: 终端节点， 不能接受其他节点通过它加入网络， 通常可以进行休眠， 以降低功耗。

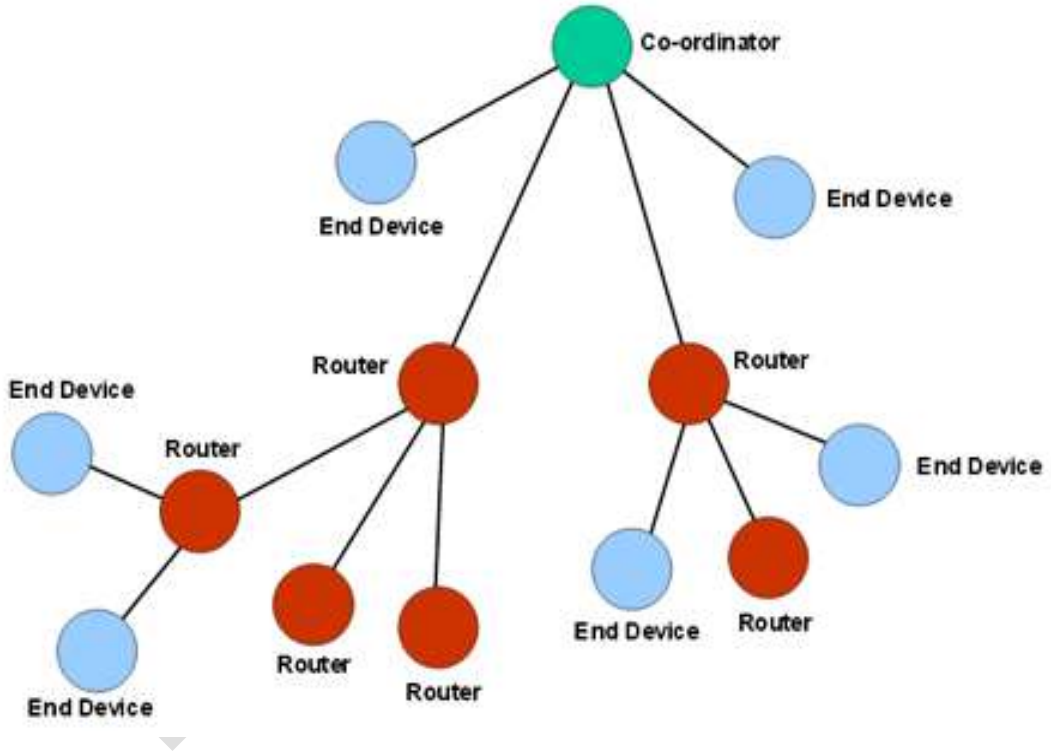
通常来说节点的类型和节点在应用中的角色是没有绝对的直接关系的。 但是用户在开发应用的时候应该根据应用的需要决定节点的类型。 比如使用电池电源， 需要休眠以降低功耗的网络节点， 通常在节点定义的时候就需要定义成为 End Device 节点。 如果整个网络中的节点都有稳定的供电， 那么除了 Coordinator 之外， 都 使用 Router 节点也是没有问题的。

无线频率和通道：

JenNet-IP 是基于 802.15.4 标准进行开发。 所以同样使用 2.4GHz 频段，2405MHz-2480MHz，一共有 16 个可用通道。 Coordinator 在创建网络的时候可以定义所使用的通道。

网络拓扑

如下图所示， JenNet-IP 网络采用的是树状网络拓扑。 Coordinator 作为网络的创建者， 在整个拓扑结构的最上层， 拥有一个或者多个 Router, End Device 节点作为 Coordinator 的子节点。 Router 节点也可以有自己的 Router, End Device 子节点。 End Device 节点没有自己的子节点。



网络标识：

就是 JenNet-IP 的 WPAN 网络区别于本网络和周围 WPAN 网络的标识数字。JenNet-IP 网络沿用的 802.15.4 网络标识 PAN ID, 为一个 16 位的数值。 同一个网络的节点具有相同的 PAN ID。 一个节点加入网络也必须获取这个网络的 PAN ID 才可以。

网络地址

在 JenNet-IP 网络中所有节点必须有自己唯一的网络地址。 JenNet-IP 网络采用了 IPv6 地址标识。 这一 128 位的地址， 高 64 位为网络标识， 低 64 位为设备标识。

Broder-Router

前文已经提到 JenNet-IP 可以方便的实现 WPAN 网络到 LAN 网络的互通， Broder-Router 就是这样一种设备， 为 WPAN 到 LAN 网络的链接提供了网关方案。 JenNet-IP 在 WPAN 和 LAN 两个领域都提供了相应的协议栈和用户开发接口。

4.2 JenNet-IP WPAN 协议栈 API 介绍

JenNet-IP 实际上包含了 WPAN 和 LAN 网络两方面的协议栈和用户开发接口。 我们在这一章只介绍 WPAN 的用户开发接口。 如果您需要了解 LAN/WAN 协议栈的相关内容， 可以参考北京博讯科技有限公司网站上的相关文档。JenNet-IP LAN/WAN Stack User Guide (JN-UG-3086)

http://www.boccn.com.cn/getfile.aspx?id=273

另外由于 JenNet-IP WPAN 协议栈的 API 众多， 这里我们只围绕核心内容介绍必要的 API。其他的详细内容请参考 JenNet-IP WPAN Stack User Guide (JN-UG-3080)

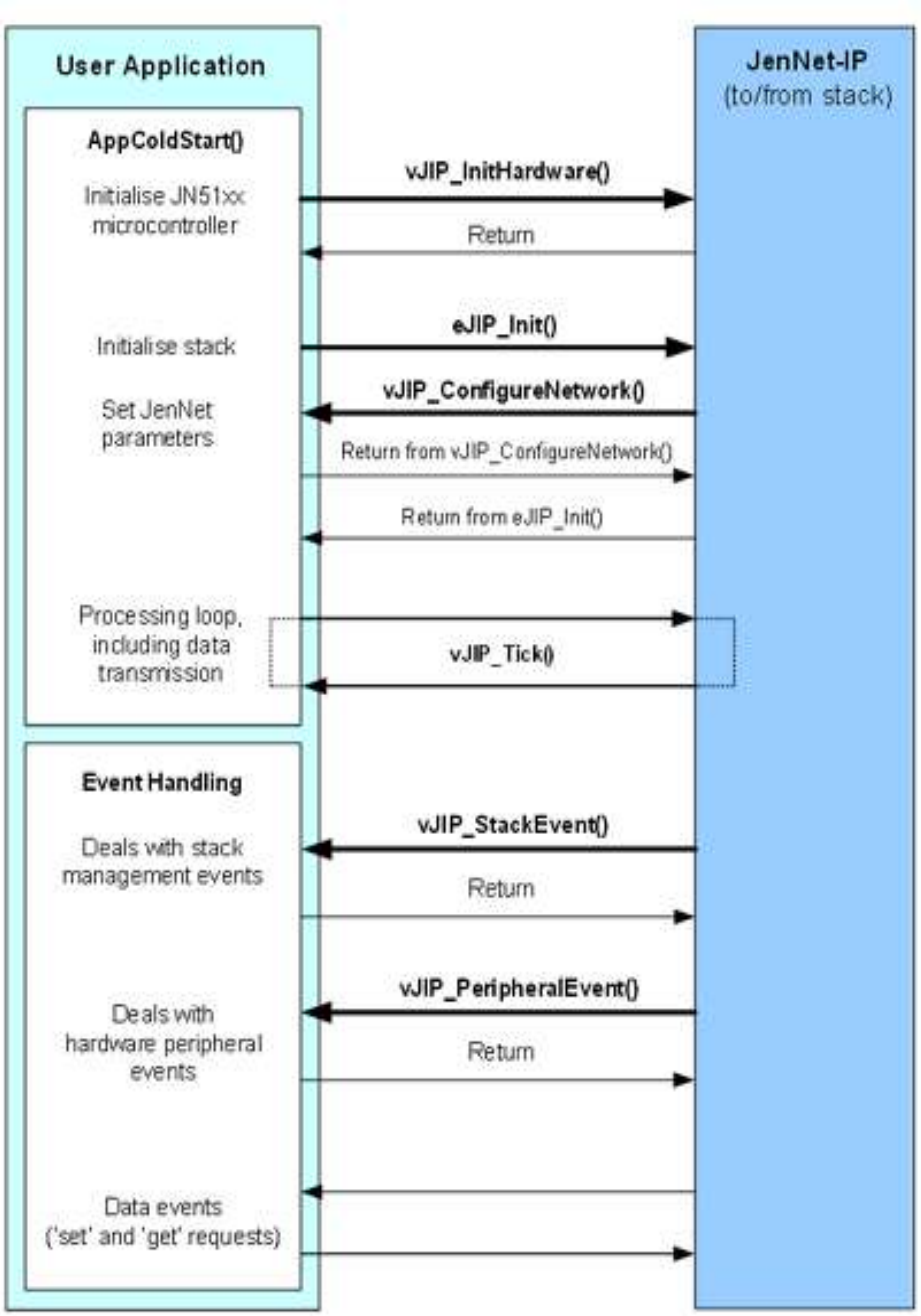
http://www.boccn.com.cn/getfile.aspx?id=271

为了理解 JenNet-IP 应用的开发。 建议大家使用我们所提供的 JenNet-IP 的模板工程。 请从博讯网站下载或者向我们的技术人员索取模板工程。

下载后的模板工程是一个命名为 JenNet-IP-AppTemplate.zip 的文件， 将这个文件解压， 把项目文件夹移动到 C:\Jennic\Application 目录下， 然后按照第二章所讲解的如何导入工程的方法将项目导入 Eclipse 的工作空间。

在 Eclipse 的 workspace 中打开\DeviceCoordinator\Source\DeviceCoordinator.c 文件。 可以参考这一文件理解 JenNet-IP 应用的大致工作原理

下图展示了 JenNet-IP WPAN 网络应用的大致流程。

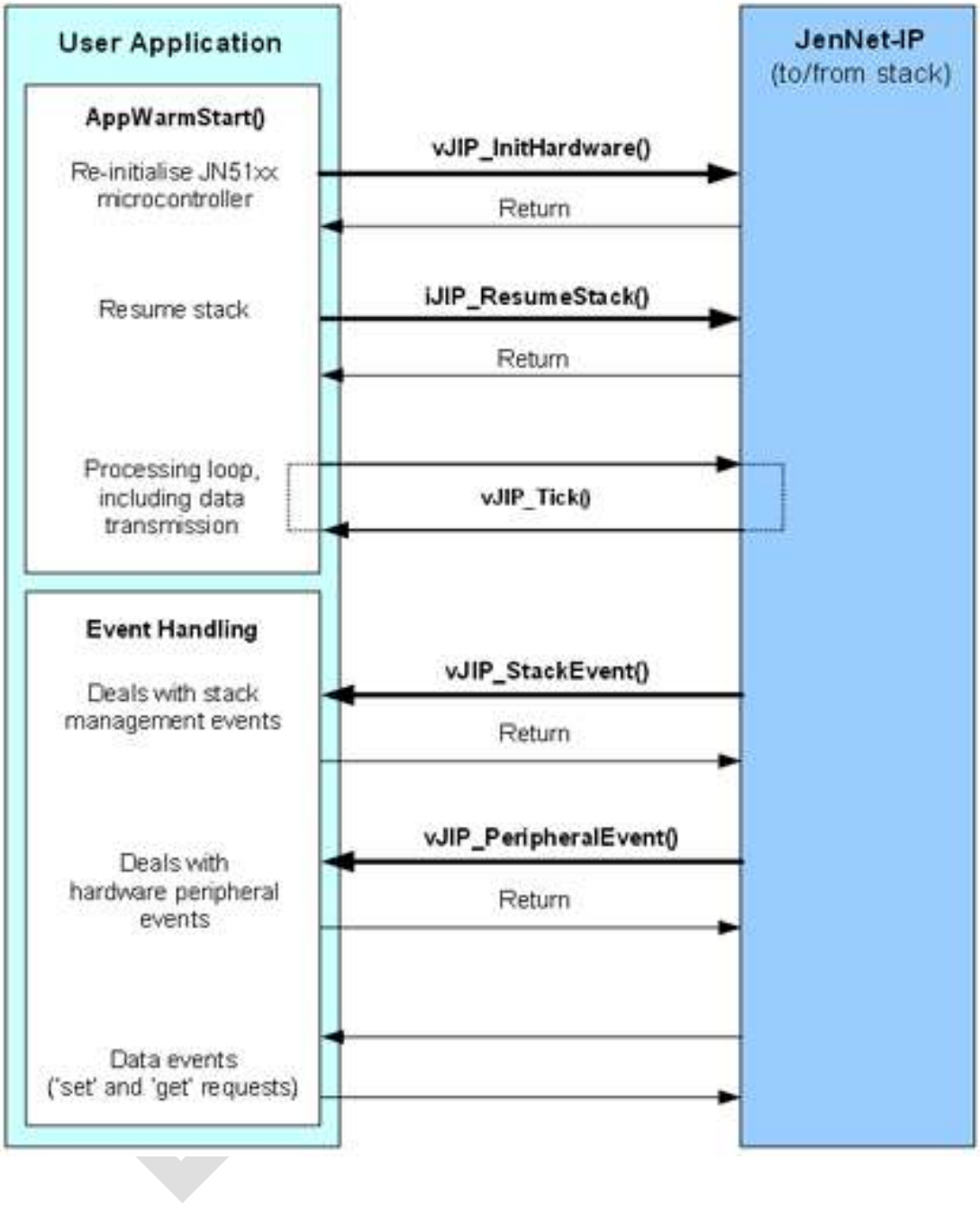


首先我们从设备启动的部分说起：

1. 首先 JenNet-IP 程序将从 AppColdStart()开始执行， AppColdStart()可以看做是一般 C 程序的 main()函数。 所有的程序都将从这里开始执行。
2. AppClodStart()将调用 v6LP_InitHardware() 进行控制器硬件的初始化。 然后调用 eJIP_Init 进行协议栈的初始化。 这里将传入一个 tsJIP_InitData 结构体作为协议栈初始化的参数。 这一结构体包含很多具体的设置， 这里只介绍几个主要的参数。
    a. u32Channel 制定需要扫描的通道

   b. u16PanId 网络 PAN ID, 如果设备是 coordinator, 那么将使用这个 PAN ID 建立网络， 如果设备是 Router 或者 End Device 这个参数将被忽略。 如果指定的是 0xFFFF 那么 Coordinator 将自动选择一个和周围网络不冲突的 PAN ID.
   c. eDeviceType 这一参数指定设备的类型是 Coordinator， Router 还是 End Device
3. 用户程序调用 eJIP_Init()函数的过程中， 协议栈会回调 v6LP_ConfigureNetwork()这个回调函数， 在这个函数中用户程序可以设置或者修改一些网络参数。 回调函数会传入 tsNetworkConfigData 结构体供用户程序进行参数的设置。 通常我们在这个回调函数中使用 v6LP_EnableSecurity()进行网络加密功能的设置。
4. 协议栈通过 eJIP_Init()初始化完成后， 程序进入主循环。 用户程序必须在主循环 while(1)中调用 v6LP_Tick() 来激发各种协议栈事件和数据事件。

以上过程描述的是设备的一次冷启动， 如果设备之前通过程序进入休眠状态， 那么热启动是从 AppWarmStart(void)开始的。 具体过程如下图。



热启动的过程基本上和冷启动大同小异。 这里不再详述。

程序运行后， 协议栈会产生各种事件， 比如外围硬件产生的中断事件， 或者各种数据传

输事件， 网络事件。 协议栈会调用相应的事件回调函数来通知用户对事件进行处理。

1. vJIP_StackEvent()。 这一回调函数用来处理各种协议栈事件。 比如网络启动， 节点加入， 或者加入网络等等。
2. v6LP_PeripheralEvent()。 外围设备中断回调函数。 用户在这个函数中可以处理外围设备产生的事件， 比如 tick timer 或者串口等等。 值得说明的是， 协议栈本身会产生一个 10ms 的 tick timer 中断。
3. v6LP_DataEvent() 发送或者接受到数据时的回调函数。
4. vJIP_Remote_DataSent() 用户程序向远程节点发送 eJIP_Remote_Mib_Set()请求时产生这一回调函数。 数据发送 成功就会调用这一回调函数。
5. vJIP_Remote_SetResponse()用户程序向远程节点发送 eJIP_Remote_Mib_Set()请求时产生这一回调函数。 远程节点接受数据， 并设置成功， 协议栈就会调用这一回调函数。

关于数据发送和接收。

JenNet-IP WPAN 使用 MIB （management information base）进行数据的发送和接收。

MIB 可以理解为定义在设备上的一组数据集合。 每一个设备节点上可以定义多个 MIB， 每一个 MIB 又可以定义多个变量。

JenNet-IP 网络上发送数据的机制就是， 节点上可以定义不同的 MIB 变量， 如果需要向远程节点传输数据， 只需要知道对方节点的网络地址还有 MIB 变量标识， 并且发送 MIB 变量的赋值请求(通过 eJIP_Remote_Mib_Set()函数)， 如果对方节点确实定义了相应的变量，那么数据就将发送成功。

对于数据的接收节点来说， 每一个 MIB 变量都可以关联定义相应的回调函数， 数据被接收到之后协议栈就将调用变量关联的回调函数， 并传入数据。 接收数据的节点就可以在回调函数中处理接收到的数据了。

MIB 变量的定义其实是稍微有一点儿复杂的。 最好是通过参考范例代码来编写自己的 MIB 还有相应的数据处理函数。 所以我们将在下两节的范例代码中结合范例程序来介绍具体的编写过程。

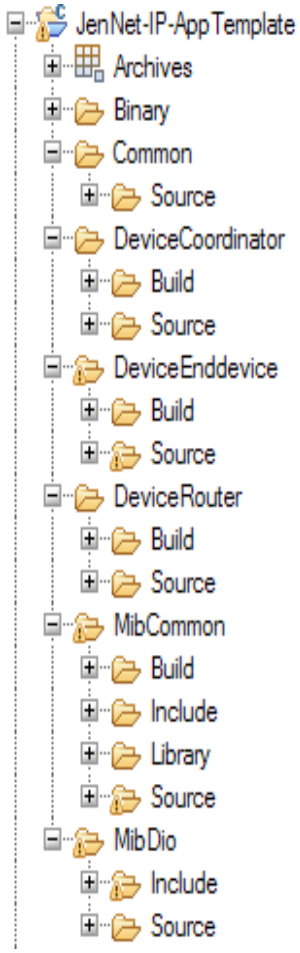
4.3 JenNet-IP 模板工程代码解释及使用

通常来说， 开发者无法通过 Eclipse 来直接建立 JenNet-IP 应用。 比较容易而且快速的做法就是使用我们所提供的 JenNet-IP 模板工程， 在模板工程的基础上开发自己的应用。

您可以从博讯的网站上下载我们所提供的 JenNet-IP 模板工程， 也可以联系我们的技术人员索取模板工程。

下载后的模板工程是一个命名为 JenNet-IP-AppTemplate.zip 的文件， 将这个文件解压， 然后给解压后的文件夹改成你的项目名称， 把这个文件夹移动到 C:\Jennic\Application 目录下， 然后按照第二章所讲解的如何导入工程的方法将项目导入 Eclipse 的工作空间。

模板工程目录内容介绍



如上图所示， 这就是导入到工作空间的模板工程的目录结构， 下面我们就解释一下每个目录的内容。

Binary 目录: 编译后的 Bin 文件的存放目录， JenNet-IP 模板工程可以编译出三种可以烧录到测试板的 bin 文件， 分别是 Coordinator, Router, EndDevice 的三个 bin 文件， 开发者可以通过编译工具按钮选择需要编译的文件。

Common 目录： 这里存放一些公用的 C 代码文件， 通常情况下这个目录的文件都不需要修改。

DeviceCoordinator 目录： 这个目录的 source 子目录存放 Coordinator 的源代码文件， 如果需要为 coordinator 设备编写功能， 就需要修改这个目录下的文件。 Build 子目录是 coordinator 的中间编译文件， 还包含编译 coordinator 的 makefile 文件， 如果需要修改编译选项就需要修改这个目录下的 makefile 文件

DeviceEnddevice 目录：这个目录的 source 子目录存放 Enddevice 的源代码文件， 如果需要为 Enddevice 设备编写功能， 就需要修改这个目录下的文件。 Build 子目录是 Enddevice 的中间编译文件， 还包含编译 Enddevice 的 makefile 文件， 如果需要修改编译选项就需要修改这个目录下的 makefile 文件

DeviceRouter 目录：这个目录的 source 子目录存放 Router 的源代码文件， 如果需要为 Router 设备编写功能， 就需要修改这个目录下的文件。 Build 子目录是 Router 的中间编译文件， 还包含编译 Router 的 makefile 文件， 如果需要修改编译选项就需要修改这个目录下的 makefile 文件

MibCommon 目录： 这个目录保存常用的一些 Mib 定义和实现文件， 通常不需要修改。

MibDio 目录： 这个目录给出了一个非常简单的 Mib 范例， 如果需要开发自己的 Mib 可以参考或者在这个范例的基础上修改。 具体的方法请参考下一节。

代码说明

下面我们来说明一下在模板工程的代码上如何编写自己的代码。

我们主要需要修改下面四个目录下面的代码

DeviceCoordinator/Source

DeviceRouter/Source

DeviceEnddevice/Source

MibDio

关于 MIB 的编写办法请参考下一小节， 这里暂时不详细解释。

下面分别解释一下 Coordinator， Router， Enddevice 的范例代码

DeviceCoordinator.c

包含了 Coordinator 的基本代码，

参考上一节的 JenNet-IP WAPN 的 API 介绍我们可以了解大部分的函数的含义。 这里需要说明的是

1. PUBLIC void Device_vInit(bool_t bWarmStart) 是设备的初始化过程， 这一过程最后进入程序的主循环。
2. PUBLIC void Node_eJipInit(void) 是设置和启动网络的过程。
3. PRIVATE void SendDataToRemote(uint8 OnorOff) 是我们给出的一个向远程节点发送数据的范例函数。 您可以在这个函数的基础上进行修改， 以完成自己需要实现的功能。
4. 因为 JenNet-IP 协议栈会自动产生一个 10ms 的 tick timer 中断， 所以我们在 v6LP_PeripheralEvent 加入了一点儿代码用来周期的调用 SendDataToRemote 向远程节点发送数据。

DeviceRouter.c

包含的 Router 的基本代码

Router 的代码和 Coordinator 的代码基本上逻辑是相同的。 所以请参考 coordinator 的代码来看。

DeviceEnddevice.c

EndDevice 的代码和上面两个设备的代码略有不同。

1. Device_vInit 最后将根据数据接收的情况进入休眠状态。
2. Device_vTick 会周期性的调用协议栈 tick
3. Device_vSleep 设定了设备休眠一秒钟
4. 因为设备被设定为 EndDevice, 而且默认开启了自动数据 poll 的机制(poll 机制就是发往 enddevice 设备的数据在设备休眠的时候会缓存在其上级节点， enddevice 唤

醒后自动从上级节点下载缓存的数据， 而且 enddevice 也会每隔 5 秒从上级节点下载缓存的数据。)

5. vJIP_StackEvent 的 E_STACK_POLL 事件展示了 poll 数据的处理机制。
6. Device_vInit 代码中加入了每次设备 warmstart 都会向远程节点发送一次数据的相应功能代码。

以上就是代码的解释， 下面说明一下模板工程的功能演示。

可以准备两块测试板， 一块作为 coordinator， 另一块作为 Router 或者 Enddevice.

将程序写入测试板。

如果是 coordinator 和 router 作为两个节点， 他们会周期性的向对方发送数据， 表现就是 router 节点的上方指示灯会闪烁， coordinator 的下方指示灯会闪烁， 而且闪烁的速率是 router 的两倍。

如果是 coordinator 和 enddevice 作为两个节点， 只有 coordinator 的两个指示灯会周期性的闪烁， enddevice 的指示灯将没有规律的闪烁， 因为其休眠和 poll 的机制， enddevice 的指示灯只会显示最后接受到的数据状态。

如果有多个节点加入网络， 那么只有 coordinator 和最后加入的节点会互相发送数据。 因为代码中只保存最后连入节点的地址。

因为模板工程设计的简单的闪灯功能， 所以用户也可以利用测试板进行通讯距离的测试。方法就是不断的移动距离观察闪灯的情况来判断数据发送接收是否正常。

如果给测试板接上编程线， 并设定 115200,8,n,1 无流控， 可以从超级终端看到调试信息。

4.4 编写 MIB 类型和变量及 回调函数

前面我们详细的解释了 JenNet-IP 模板工程代码的使用方法。 下面我们将详细的解释如何定义自己的 MIB 变量。

因为 MIB 的变量定义稍微有点儿复杂， 所以不建议开发人员从头进行定义。 您可以利用我们提供的模板工程代码， 在其基础上进行 MIB 变量的添加。

我们提供的模板工程带有一个简单的 MIB 定义。 您可以在 Eclipse 中打开工程的 MibDio 目录，这一目录包含 Include 和 Source 两个目录。

Include 包含的是头文件， 分别是

MibDio.h 定义了 MIB 的 ID 常量和变量的 ID 常量

MibDioControl.h 定义了 MIB 使用的数据结构还有 MIB 初始化， 注册 以及各回调函数

MibDioControlDef.h 是 MIB 的定义文件

Source 目录包含的是实现文件。

MibDioControl.c MIB 初始化， 注册以及变量回调函数的实现

MibDioControlDec.c 关联各 MIB 变量和对应的回调函数

下面我将逐步的讲解如何添加一个我们自己的 MIB 变量。 这里假设我们需要添加一个 ID 为 `VAR_IX_USER_DATA` 的变量，数据类型是 `uint32`， 我们还将为其添加相应的回调函数用来处理接收到的数据。

第一步：首先从定义常量开始， 打开 MibDio.h 文件。 在相应的位置添加

```
#define VAR_IX_USER_DATA        3
```

第二步：在 MibDioControlDef.h 文件中添加 这一变量的定义

```
DEFINE_VAR(VAR_IX_USER_DATA,                UINT32, UserData, NULL, 0,  (READ | WRITE | TRAP), NONE, NONE)
```

第三步：MibDioControl.h 中，tsMibDioControlTemp 结构体中加入 uint32 u32UserData;

并且定义回调函数 PUBLIC teJIP_Status MibDioControl_eSetUserData (uint32 u32Val, void *pvCbData);

第四步：在 MibDioControlDec.c 文件中定义回调函数和 MIB 变量的关联。
JIP_CALLBACK(UserData, MibDioControl_eSetUserData, vGetUint32, &sMibDioControl.sTemp.u32UserData)

第五步： 在 MibDioControl.c 文件中定义具体的回调函数实现。

```
PUBLIC teJIP_Status MibDioControl_eSetUserData(uint32 u32Val, void *pvCbData)
{
   teJIP_Status   eReturn = E_JIP_OK;

         /* Write new value */
   DBG_vPrintf(MIB_DIO_CONTROL_DBG, "\r\n data=%c", (uint8)u32Val);

         vJIP_NotifyChanged(psMibDioControl->hMib, VAR_IX_USER_DATA);

   return eReturn;
}
```

这样我们就完整的定义了一个 MIB 变量， 可以用来接收其他节点通过 eJIP_Remote_Mib_Set()函数发送来的数据， 并且在回调函数中进行数据的处理

4.5 MIB Traps

JenNet-IP 协议栈实现了类似 SNMP 的 Traps 机制用来进行对于 MIB 变量的监测读取。

Traps 机制简单的解释来说就是， 被监测的 MIB 变量在变化的时候将向所有曾经向这一变量注册过的节点发出通知， 这样所有曾经注册的节点都可以知道 MIB 变量的变化。

基于 JenNet-IP 实现 Traps 需要在本地节点（数据的持有节点）和远程节点（对数据感兴趣的节点， 既数据的接收节点）进行一系列操作。

本地节点（数据的持有节点）

首先本地节点需要已经注册了相应的 MIB 及其变量， 而且在注册 MIB 变量的时候需要标记其可以被 Trap.

然后当本地节点对变量进行了修改， 并且认为有必要通知对这一变量感兴趣的节点的时候，需要调用 vJIP_NotifyChanged()进行通知。

远程节点 （对数据感兴趣的节点）

首先对数据感兴趣的节点需要使用 eJIP_Remote_Trap()向数据的持有节点进行 Trap 注册，这一过程就是告知数据的持有节点，一旦数据发生变化， 请通知我。 这一函数将返回两个回调函数。

回调函数：vJIP_Remote_DataSent() 当 Trap 请求发送成功时返回这个回调函数

回调函数： vJIP_Remote_TrapResponse()当 Trap 请求被数据持有节点处理之后返回这个回调函数， 返回的参数标记了 Trap 请求是否被接受。

当数据变化通知到达时， 远程节点会收到 vJIP_Remote_TrapNotify()回调函数， 这一回调函数的参数带有变化后的变量值。

远程节点可以通过 eJIP_Remote_Untrap()取消对变量的 Trap.

需要注意的是： 当本地节点或者远程节点任何一方重启， 那么原本注册的 Trap 都将丢失，需要重新注册。

范例程序：

我们提供了基于 JenNet-IP 的 Trap 范例程序来演示如何使用 Trap 机制。 您可以解压 JenNet-IP-WSN-Trap.zip 文件，并将其导入 Eclipse 来参考其代码。

下面对范例程序的代码进行一些解释。

DeviceRouter.c 作为本地节点， 是 MIB 变量的持有方。

v6LP_PeripheralEvent 过程中利用 Tick Timer 周期性的修改 MIB 变量 VAR_IX_DIO_CONTROL_OUTPUT_ON，并进行通知。

这里我们为了演示， 对于这个变量所携带的数据进行了一些处理。 首先我们利用 u32AHI_DioReadInput()函数模拟一个发送者的 ID， 这样用户可以利用测试板的拨码开关模拟不同的发送者 ID， 以便用户可以使用多个测试板来进行测试。

然后我们在演示程序中设定了一个不断增加的 msgIndex 作为数据， 这样可以在数据接收方观察数据的丢包情况。

所以 VAR_IX_DIO_CONTROL_OUTPUT_ON 变量实际上携带了以上两个数据。 我们在接收方会对这个数据进行解析。

DeviceCoordinator.c 作为远程节点。

首先我们在 vJIP_StackEvent 过程中的 E_STACK_NODE_JOINED 事件调用了 QueryRemoteMibIndex()过程对新加入的节点进行了 MIB Index 的查询。 这一过程主要是为了获取我们感兴趣的 MIB 的索引号， 取得了这一索引号我们才可以向本地节点进行 Trap 的注册。

vJIP_Remote_QueryMibResponse 过程通过对比 MIB ID 取得了所需要的 MIB Index. 然后这一过程调用 RegisterRemoteTrap()向本地节点注册 Trap.

vJIP_Remote_TrapResponse 回调函数返回 Trap 是否注册成功。 如果没有注册成功将返回各种错误代码

vJIP_Remote_TrapNotify 回调函数用来接收变量的更新通知， 这一回调函数的参数带有变量的变化后的值。 范例程序将变量值进行解析， 并通过串口输出。

您可以使用超级终端（配置：115200-8-n-1 无流控）连接 coordinator 节点观察所 Trap 的数据不断的发送到 coordiantor。 您也可以使用多个 Router 节点， 并通过测试板的拨码开关模拟不同的 ID， 观察多个节点同时向 coordinator 发送数据的情况。

5：外围部件的操作

UART

JN516x 微控制器有两个 Uart. 分别是 Uart 0 和 Uart 1.

Uart0 可以采用 4 线模式 (TxD, RxD, RTS, CTS) 或者 2 线模式(TxD, RxD)

Uart1 可以采用 2 线模式（TxD, RxD）或者 1 线模式(TxD)

两个 Uart 都和 DIO 共用。

信号线	Uart 0	Uart 1
CTS	DIO4	
RTS	DIO5	
TXD	DIO6	DIO14
RXD	DIO7	DIO15

可以使用 vAHI_UartSetLocation()函数将 Uart 0 的信号线配置到 DIO12-DIO15

或者将 Uart1 的信号线配置到 DIO11 和 DIO19

JN516x 的 Uart 带有 FIFO 缓存， 大小从 16 bytes 到 2047bytes 可配置。

下面介绍一下 Uart 的配置和使用的相关函数， 这里只介绍常用的 Uart 函数， 更详细的说明请参考 JN-UG-3087 的外围部件说明手册。

bAHI_UartEnable()： 将使相应的 Uart 可用， 当相应的 Uart 使能后， 其共用 DIO 将失去 IO 功能

vAHI_UartSetRTSCTS() : 将 Uart0 配置为 2 线模式， 不使用流控信号线

bAHI_UartTxOnly(): 将 Uart1 配置为 1 线模式， 不使用 RxD, 这种模式下 Uart1 只能发送数据。

vAHI_UartSetBaudRate(): 用于设置串口波特率.

vAHI_UartSetControl(): 用于设置串口的其他配置参数， 比如奇偶校验， 数据位， 停止位。

vAHI_UartSetInterrupt(): 用于设置串口中断， 可设置是否接收各种串口中断事件， 比如 FIFO empty, 数据事件， CTS 事件等等。

vAHI_Uart0RegisterCallBack(), vAHI_Uart1RegisterCallback(): 这两个函数分别用于设置接收串口事件的回调函数。 对于使用 802.15.4 或者 JenNet-IP 工程模板的开发者， 因为开发模板工程都带有了相应的统一的回调函数， 所以开发者不需要自己注册回调函数。 后面我们将由例子进行说明

vAHI_UartWriteData(): 向串口写入数据。 数据将会被缓存到 FIFO, DMA engine 将会自动在串口可以写数据的时候将数据写出。

u8AHI_UartReadData(): 从串口读取数据。 通常我们会在回调函数中读取数据。 所以这个函数不是必须的。

关于流控：

对于 Uart0， 如果采用 4 线模式， 那么可以自己通过读取 CTS, RTS 的状态来自己控制流控，具体的方法可以参考 JN-UG-3087 的 6.5.2 小节。

但是我们推荐使用自动流控的方式。

外围部件 API 提供了 vAHI_UartSetAutoFlowCtrol()函数用来设置自动流控。

范例程序：

我们随文档提供了一个读写 Uart 的演示程序。JN-Demo-Uart.zip.

开发者可以将这个程序解压， 导入 Eclipse 的 workspace.

下面对这个演示程序进行说明

1. 程序采用 802.15.4 的工程模板（JN-AN-1174）作为基础.
2. 程序的 Coordinator 代码演示了串口的基本读写。请打开 AN1174_154_Coord.c 阅读其代码。
3. 程序使用 Uart0 进行读写操作， 连接编程电缆后， 串口设置 19200, 8,n,1 无流控。通过超级终端将会看到串口持续的写出单个字母， 如果通过 PC 向其输入字母，串口会返回最后一个输入的字母。

如何使用其代码到自己的项目：

第一步：范例程序使用了一个通用的功能文件 Printf.c 用来向串口输出数据。 首先将这一文件从范例程序的 common/source 目录拷贝到自己的项目。

第二步：修改 makefile, 在 Application Source files 部分加入 APPSRC += Printf.c

第三步： 在需要使用串口输出的文件加入

```
PRIVATE void vPutChar(unsigned char c);
PRIVATE void vUartInit(void);
```

两个函数的声明

及其相应的函数实现

```
PRIVATE void vPutChar (unsigned char c)
{
  while (!(u8AHI_UartReadLineStatus(UART) & E_AHI_UART_LS_THRE));
  vAHI_UartWriteData(UART, c);
  while ((u8AHI_UartReadLineStatus(UART) & (E_AHI_UART_LS_THRE | E_AHI_UART_LS_TEMT))
       != (E_AHI_UART_LS_THRE | E_AHI_UART_LS_TEMT));


}

PRIVATE void vUartInit (void)
{
#if (UART == E_AHI_UART_0)
  vAHI_UartSetRTSCTS(UART, FALSE);                    /* Disable use of CTS/RTS      */
#endif
  vAHI_UartEnable(UART);
  vAHI_UartReset(UART,                                /* 复位收发 FIFO              */
           TRUE,
           TRUE);
  vAHI_UartSetBaudRate(UART, E_AHI_UART_RATE_19200);          /* 设置波特率
*/
  vAHI_UartSetInterrupt(UART, TRUE, FALSE, TRUE, TRUE, E_AHI_UART_FIFO_LEVEL_1);  /*  打
开中断 */
  vInitPrintf((void *)vPutChar);
}
```

第四步： 定义需要使用的串口

```
#define UART              E_AHI_UART_0
```

第五步： 在程序启动阶段 调用 vUartInit(); 初始化串口
第六步：在需要向串口输出数据的时候使用 vPrintf();
第七步： 可以在硬件外围回调函数中处理串口输入数据

```
PRIVATE void vProcessIncomingHwEvent(AppQApiHwInd_s *psAHI_Ind)
{

        if (psAHI_Ind->u32DeviceId == E_AHI_DEVICE_UART0)
                {
                        /* If data has been received */
                        if ((psAHI_Ind->u32ItemBitmap & 0x000000FF) ==
E_AHI_UART_INT_RXDATA)
                        {
                                cCharIn = (uint8)((psAHI_Ind->u32ItemBitmap & 0x0000FF00) >>
8);

                        }

                }


}
```

DIO / DO

JN516x 具有 20 路可配置的 DIO, 和两路 DO.

20 路可配置的 DIO 可以自由灵活的配置为输入或者输出， 同时也和其他的外围接口共用，比如 ADC, UART, Timer, SI, SPI 等等。 当其他外围部件的接口通过相应的函数激活为其功能后， DIO 就会失去 IO 的功能。

配置 DIO:

可以通过 vAHI_DioSetDirection 函数来配置 DIO 的输入输出。 其参数为 32 位的 bitmap，参数的 0-19 位， 对应 DIO 的 0-19 位. 如果想要设置哪个位置， 只要相应的位为 1 就可以了。

比如想要配置 DIO9 为输出的话 （博讯测试板的上指示灯）， 就可以使用如下语句进行设置

vAHI_DioSetDirection(0,0x200);

其中 0x200 = 二进制的 1000000000， 所以就是设置 DIO9 为输出。

设置输出：

如果设置输出就可以使用 vAHI_DioSetOutput()来向 DIO 进行输出。

vAHI_DioSetOutput(0x200,0); 设置 DIO9 为 on,，测试板上指示灯灭

vAHI_DioSetOutput(0,0x200); 设置 DIO9 为 off， 测试板上指示灯亮

读取 DIO:

可以使用 u32AHI_DioReadInput()读取全部的 DIO 状态。

也可以使用 u8AHI_DioReadByte 读取一个 byte。

注册中断回调函数

可以使用

vAHI_DioInterruptEdge 来设置 DIO 的上升沿还是下降沿触发中断，

vAHI_DioInterruptEnable 来使能中断

可以使用 vAHI_SysCtrlRegisterCallback 来注册 DIO 的中断回调函数

回调函数的原型必须是

```
PRIVATE void CallBackFunction(uint32 u32Device, uint32 u32ItemBitmap)
```

注册之后可以使用 u32ItemBitmap 来接收特定的 DIO 事件。 比如

```
if (E_AHI_DIO8_INT & u32ItemBitmap)
{
        //do something
}
```

这样就是接收 DIO8 的事件

设置 DIO 为唤醒输入

方法和设置中断是基本相同的， 只不过使用的函数是

vAHI_DioWakeEdge() 用来设置 上升沿还是下降沿唤醒

vAHI_DioWakeEnable() 用来使能 DIO 唤醒。

使用 DO.

JN516x 具有两路 DO, DO0 和 DO1. 使用非常简单， 使能之后， 只能用于输出

bAHI_DoEnableOutputs 用来使能 DO

vAHI_DoSetDataOut 用来设置输出

vAHI_DoSetPullup() 用来设置 pullup 状态为 on 或者 off

TickTimer

Tick Timer 可以产生周期性的中断。 可以用于下列功能的实现：

1. 产生周期性的中断
2. 执行周期性的任务
3. 计时
4. 系统的监控， 比如和 watchdog 一起使用。

Tick Timer 是采用计数的方式， 当达到预设值时根据工作模式的不同会执行下列操作中的一种

1. 继续向上计数
2. 复位计数为 0， 并重新开始向上计数
3. 停止计数

当到达设定的参考计数值时， 可以根据设定产生一个中断。

如何使用 Tick Timer

通常我们会通过下面的几个 API 函数来配置使用 TickTimer

vAHI_TickTimerConfigure()函数， 这一函数用来指定 tick timer 是否工作， 还有工作的模式

其参数有四个选择

E_AHI_TICK_TIMER_CONT ： 到达参考值之后继续计数

E_AHI_TICK_TIMER_RESTART： 到达参考值之后从 0 开始计数

E_AHI_TICK_TIMER_STOP： 到达参考值之后停止计数。

E_AHI_TICK_TIMER_DISABLE： 不使用 Tick Timer

vAHI_TickTimerWrite() 函数， 这一函数将设定 tick timer 的起始计数， 通常我们都设置为 0

vAHI_TickTimerInterval()函数， 设置 tick timer 的参考计数， 通常计算方法如下。

TICK_PERIOD_COUNT（计数值） = 16000 * TICK_PERIOD_ms （计数毫秒数）

比如如果你想设定一个周期为 10ms 的 tick timer， 那么 设定的计数值就应该是

TICK_PERIOD_COUNT = 16000 * 10 = 160000

vAHI_TickTimerIntEnable() 函数用于让 tick timer 在计数值达到参考值的时候产生中断。

vAHI_TickTimerInit() 用于注册 tick timer 的中断回调函数， 也可以使用 vAHI_TickTimerRegisterCallback()， 作用是一样的。

所注册的回调函数的原型应该为：

PRIVATE void vTickTimerISR(uint32 u32Device, uint32 u32ItemBitmap)

需要说明的是， 如果你使用的是 JenNet-IP 的应用模板。 那么 JenNet-IP 的协议栈本身就会自动产生一个 10ms 的 tick timer。 所以只需要在模板留有的外围部件的回调函数处理这一中断即可。

代码如下

```
PUBLIC void v6LP_PeripheralEvent(uint32 u32Device, uint32 u32ItemBitmap)
{

        if (u32Device == E_AHI_DEVICE_TICK_TIMER)
        {
                //deal with tick timer event
        }
}
```

范例代码：

我们随本开发指南提供了一个如何使用 tick timer 的 Demo. 用户可以解压 JN-Demo-TickTimer .zip 文件， 并导入 Eclipse 中查看代码。

这一范例使用 802.15.4 的模板工程， 并添加了 tick timer 使测试板实现闪灯的效果。

vInitSystem 函数的后部初始化了 Tick Timer，设定为 10ms 的周期， 并注册了回调函数 vTickTimerISR。 具体代码及注释如下：

```
/* Initialise tick timer to give 10ms interrupt */
vAHI_TickTimerConfigure(E_AHI_TICK_TIMER_DISABLE); //关闭 tick timer
vAHI_TickTimerWrite(0); //设置初始计数值
vAHI_TickTimerInit(vTickTimerISR); //注册中断回调函数
vAHI_TickTimerInterval(TICK_PERIOD_COUNT);//设置参考计数值
vAHI_TickTimerConfigure(E_AHI_TICK_TIMER_RESTART);//使能 tick timer,并设置为重复计数模式
vAHI_TickTimerIntEnable(TRUE); //使能中断回调函数
```

vTickTimerISR 中实现了对 DIO9（测试板的上指示灯）的闪灯功能。

```
PRIVATE void vTickTimerISR(uint32 u32Device, uint32 u32ItemBitmap)
{

        static uint8 u8ToggleCount;

  /* Flash LED 1 to show we are alive */
  if (u8ToggleCount++ & 0x20)
  {
```

```
        vAHI_DioSetOutput(0x200,0 );


    }
    else
    {
        vAHI_DioSetOutput(0,0x200 );


    }

}
```

Flash

本节将介绍如何使用外围部件 API 来操作 Flash 存储数据。 Flash 可以作为非易失性内存在系统完全没有电源的时候保存数据。 可以用来保存系统配置等。

JN516x 的 Flash 存储是分割成不同的 Sector 的， 不同的型号具有不同的 sector 数量。 另外 JN516x 还可以使用外部 Flash 作为存储, 具体可以使用的 Flash 芯片型号可以参考 JN-UG-3087。 本节我们只重点介绍如何使用内部 Flash.

芯片型号	Sector 数量	Sector 大小(K Bytes)	总大小 （K Bytes）
JN5168	8	32	256
JN5164	5	32	160
JN516x	2	32	64

如表格所示， JN5168 具有 8 个 sector, 可以理解为 JN5168 的 Flash 被分为 8 个区段， 编号为从 0-7， 每个区段的大小是 32Kbytes, 所以 JN5168 的总 Flash 大小就是 256Kbytes.

通常系统使用从 0 开始的 Sector 来存储应用程序本身， 所以开发者通常应该使用后面的 Sector 来存储数据， 比如对于 JN5168 来说可以使用 Sector 7 来存储数据。

Flash 的初始化：

bAHI_FlashInit(); 函数用来初始化 Flash, 这一函数需要两个参数， 第一个参数是 Flash 的类型， 如果我们使用内部 Flash 就可以传入 E_FL_CHIP_INTERNAL， 第二个参数是操作 Flash 的函数表， 如果我们使用内部 Flash , 这一参数就可以传入 NULL.

Flash 的擦除：

Flash 的写入原理是只将必要位的 1 修改为 0， 所以任何一次写入之前都需要将整个 Sector 进行一次擦除操作， 擦除操作就是将一个 sector 所有的位都置为 1， 为后续的写入操作做准备。

bAHI_FlashEraseSector（）函数用来进行擦除操作， 传入的参数是 sector 的编号， 对于 JN5168 来说就是 0-7. 但是需要注意， 一定不要试图擦除前面的几个 sector, 因为从 0 开始的 sector 是存放用户程序的区块。 如果擦除将导致系统无法运行。

Flash 数据的写入：

Flash 数据的写入稍微有点儿复杂， 使用 bAHI_FullFlashProgram()可以写入数据到任何区块的地址。

这一函数的原型是

bool_t bAHI_FullFlashProgram(

uint32 u32Addr

,

uint16 u16Len

,

uint8 *pu8Data

);

下面对三个参数进行具体的解释。

u32Addr, 为写入数据的偏移地址， 因为这一地址是整体寻址的， 所以需要按照写入的区块进行计算。

偏移地址 = 每个区块大小(单位 KB) * 1024 * 区块编号 + 区块内偏移地址

比如我们需要写入数据到 sector 7. 就是第八个区块的第 0 个地址。 那么

偏移地址 = 32 * 1024 * 7 +0

u16Len 为写入数据的大小， 这里需要注意， 我们每次需要写入的数据只能是 16 个字节的整数倍， 因为 Flash 是按照 pagewords 进行操作的， 每个 pageword 的大小就是 16 个字节，所以每次写入数据的大小也只能是 16 的整数倍字节。

pu8Data 写入数据的起始地址。 仍然需要注意定义的数据大小应该是 16 字节的整数倍。通常我们会定义一个结构体来传递数据， 所定义的结构体需要考虑到大小应该是 16 字节的整数倍。

因为 Flash 是按照 sector 来进行写入操作， 所以正确的写入流程如下

1. 读取 sector 的数据到 RAM
2. 擦除整个 sector
3. 修改 RAM 中的数据
4. 将 RAM 中数据写回到 Flash 的 sector

通常来说 Flash 都有一定的擦除写入寿命， 大概是 10000 个循环左右。 所以对于 Flash 的写入操作不应过于频繁。 请开发者注意合理的设计系统以降低 Flash 的擦写操作。

读取 Flash 数据：

bAHI_FullFlashRead（）函数用来读取 Flash 中的数据， 这一函数的三个参数和写入 Flash 数据函数的参数是一样的， 说明也完全一样， 请参考上文来理解。 这里不再详述。

Flash 操作 Demo 程序：

我们随手册提供了如何操作内部 Flash 的 Demo 程序， 请将 JN-Demo-Flash.zip 解压并导入到 Eclipse 的 workspace 中。 这里对其代码进行简单说明。

这一 Demo 程序使用 802.15.4 的模板工程作为基础， 演示了内部 Flash 的读取写入数据的方法。 开发者可以使用测试板写入程序， 然后连接串口通过超级终端看到程序将接收超级终端输入的字符， 并且将字符写入 Flash， 程序周期性的读取 Flash 数据并且显示在超级终端上。（超级终端连接设置为 115200, 8-n-1 无流控）

首先定义了常量， 使用编号为 7 的 sector 来保存数据， 并且偏移地址为 sector 的 0 号地址

```
#define FLASH_SECTOR_NUM        7

#define FLASH_OFFSET_ADDR       32 * 1024 * FLASH_SECTOR_NUM +0
```

vInitSystem 过程中初始化内部 Flash

```
bAHI_FlashInit(E_FL_CHIP_INTERNAL, NULL);
```

在 Tick Timer 的 vTickTimerISR 周期回调函数中读取 Flash 数据并通过串口显示：

```
uint8 u8LoadData[16];

bAHI_FullFlashRead(FLASH_OFFSET_ADDR , 16, u8LoadData);

vPrintf("Data from Flash u8LoadData[0]= %c \n" , u8LoadData[0]);

vPrintf("Data from Flash u8LoadData[15]= %c \n" , u8LoadData[15]);
```

在硬件中断的回调函数中， 程序将读取串口输入， 并且将输入的字符写入 Flash

```
PRIVATE void vProcessIncomingHwEvent(AppQApiHwInd_s *psAHI_Ind)

{


        if (psAHI_Ind->u32DeviceId == E_AHI_DEVICE_UART0)

                {

                        /* If data has been received */

                        if ((psAHI_Ind->u32ItemBitmap & 0x000000FF) ==
E_AHI_UART_INT_RXDATA)

                        {

                                cCharIn = (uint8)((psAHI_Ind->u32ItemBitmap & 0x0000FF00) >>
8);
```

```
                        uint8 u8SaveData[16];
                        u8SaveData[0]=cCharIn;
                        u8SaveData[15]=cCharIn;

                        vPrintf("Try to write data to flash \n");
                        if (bAHI_FlashEraseSector(FLASH_SECTOR_NUM))
                        {
                                erasesuccess=1;
                        }
                        if (bAHI_FullFlashProgram(FLASH_OFFSET_ADDR , 16,
u8SaveData))
                        {
                                writesuccess=1;
                        }
                    vPrintf("erase success= %d \n" , erasesuccess);
                    vPrintf("write success= %d \n" , writesuccess);
                    }

                }

}
```

WakeTimer

JN516x 有两个唤醒计时器， Wake Timer 0 和 Wake Timer 1， 每个都是 41 位的计数器。 唤醒计时器是基于 32kHz 时钟， 可以在设备处于休眠状态时运行（前提是 CPU 必须处于运行状态）。唤醒计时器负责对睡眠时间进行计时， 并且在睡眠时间到达设定时间时产生中断， 唤醒设备。

使用唤醒计时器

vAHI_WakeTimerEnable() 函数用来指定使用的唤醒计时器。 这一函数可以使指定的唤醒计时器使能或关闭。

唤醒计时器中断属于系统中断， 所以需要使用 vAHI_SysCtrlRegisterCallback()函数注册中断回调函数。 中断回调函数对于唤醒计时器来说不是必须使用的。

vAHI_WakeTimerStartLarge() 函数用来设定唤醒计时器的计数设定值。 设定计时器的计数值之后， Wake Timer 开始倒数计数， 当计数到达 0 时将触发唤醒中断， 设备将从休眠状态被唤醒。 而且计数将自动被重新设定为 0x1FFFFFFFFF, 并继续计数。

如何根据需要休眠的时间进行计数计算：

因为唤醒计时器采用内部 32kHz 的时钟进行计数， 所以会有 18%快或慢的误差。 所以在进行计算之前必须要通过 u32AHI_WakeTimerCalibrate()函数获取矫正参数。 然后将矫正参数代入计算公式进行具体时间的计算。

而且因为矫正参数的获得是通过 Wake Timer 0 的测试运行得到， 所以获取矫正参数的过程必须在调用 Wake Timer0 之前运行。

时间计算公式如下

```
CpuSleepCount = (uint32)(32 * 1000 * CpuSleepSeconds * 10000) /
u32AHI_WakeTimerCalibrate();
```

CpuSleepSeconds 为需要休眠的时间， 单位为 （秒）

CpuSleepCount 为得到的设定计数值

vAHI_WakeTimerStop() 用于停止 wake timer 的计数。 计数值将保持当时的值， 并且 wake timer 将不再产生中断。

u64AHI_WakeTimerReadLarge() 函数可以读取 wake timer 的当前计数值。 这一函数不会停止 wake timer 的计数。

演示程序：

开发者可以通过我们随教程所附的演示程序来了解 wake timer 的用法。 将 JN-Demo-WakeTimer.zip 解压， 并导入 Eclipse 的 workspace。打开程序 AN1174_154_Coord.c 文件。

可以在

PUBLIC void AppColdStart(void)

过程中看到 wake timer 的用法。

```
CpuSleepCount = (uint32)(32 * 1000 * CpuSleepSeconds * 10000) /
u32AHI_WakeTimerCalibrate(); //校正后的计时时间， 32 为 计时时钟， 1000 为 K, 10000
为矫正参数。

  vAHI_WakeTimerEnable(E_AHI_WAKE_TIMER_0, TRUE);            /* 开中断
*/

  vAHI_WakeTimerStartLarge(E_AHI_WAKE_TIMER_0, CpuSleepCount);


  while (1)

  {

    vProcessEventQueues();




    vAHI_CpuDoze(); //CPU 时钟被暂停， 直到 WakeTimer 产生中断
```

```
(bLed = !bLed) ? vAHI_DioSetOutput(0x200,0 ) : vAHI_DioSetOutput(0,0x200 );
    vAHI_WakeTimerStartLarge(E_AHI_WAKE_TIMER_0, CpuSleepCount);
}
```

程序首先通过获取矫正参数， 计算出正确的计数值。

然后使用 Wake Timer 0 开始计数。

设定 Wake Timer 0 的计数值。

程序进入主循环。

暂停 CPU 时钟， 等待 Wake Timer 产生中断。

Wake Timer 产生中断将唤醒 CPU， 程序将从 vAHI_CpuDoze(); 这一语句继续向下执行。

程序唤醒之后将对测试板的指示灯设定状态， 以表示其运行状态。

然后重新设定 Wake Timer 的计数值， 重新开始计数。

程序进入下一个循环。

ADC

JN516x 集成 10bit ADC。 支持 6 个输入通道。 ADC1-4 可以使用外部输入， 也可以采样内部集成的温度传感器和电源电压。 具体可参考 JN516x 的 DataSheet.

使用 ADC， 这一主题涉及的内容很多， 所以我们在这个入门教程中将不做更加详细的讲解。 具体请参考 JN-UG-3087 文档的第一部分， 第四小节内容。

在这一小节我们只讲解 ADC 的基本操作和必要的 API 函数说明， 并在最后给出可以运行于测试板的例子。

vAHI_ApConfigure（）函数用来设置使用模拟量外设。 这一函数有五个输入参数分别是

bAPRegulator: 用来设定是否使用模拟量外设

bIntEnable 用来设定是否在 ADC 转换完成产生中断

u8SampleSelect 用来设定采样的时钟周期

u8ClockDivRatio 用来设定时钟

bRefSelect 用来设定使用外部参考电压还是内部参考电压

bAHI_APRegulatorEnabled()用来返回模拟量设备是否已经准备好。

vAHI_AdcEnable（）用来使能 ADC 输入。 这一函数有三个参数， 本别是

bContinuous: 用来设定 ADC 的转换模式， 是连续转换， 还是单次转换

bInputRange: 用来设定输入电压范围

u8Source: 用来选择输入源， 可选 ADC1, ADC2, ADC3, ADC4, 内置温度传感器， 供电电压

vAHI_AdcStartSample() 函数用来开始 ADC 采样。

bAHI_AdcPoll() 函数用来返回 ADC 采样和转换是否完成。

u16AHI_AdcRead() 函数用来读取 ADC 的值。

范例程序：

我们随本教程提供了 ADC 的使用例程。 用户可以将 JN-DEMO-ADC.zip 项目解压， 并导入 Eclipse。

AN1174_154_Coord.c 文件演示了如何通过 ADC 读取内部电压并通过串口输出。

vInitSystem(void) 过程的后部进行了 ADC 的设置和初始化

```
vAHI_ApConfigure(E_AHI_AP_REGULATOR_ENABLE,    //使用 AP
          E_AHI_AP_INT_DISABLE,  //关闭中断
          E_AHI_AP_SAMPLE_2, //设定采样周期
          E_AHI_AP_CLOCKDIV_500KHZ, //设定时钟
          E_AHI_AP_INTREF); //使用内部参考电压
        while (!bAHI_APRegulatorEnabled());
        vAHI_AdcEnable(E_AHI_ADC_SINGLE_SHOT,  //单次采样
          E_AHI_AP_INPUT_RANGE_2, //两倍参考电压
```

```
        E_AHI_ADC_SRC_VOLT); //选择输入来源为电源电压

    vAHI_AdcStartSample();
```

在 vTickTimerISR 过程中我们将周期的读取并转换 ADC 的值。 并通过串口显示。

```
while (bAHI_AdcPoll());

    u16AdcReading = u16AHI_AdcRead();

    /* Input range is 0 to 2.4V. ADC has full scale range of 10 bits.

    Therefore a 1 bit change represents a voltage of approx 2344uV */

    u16BattLevelmV = ((uint32)((uint32)(u16AdcReading * 2344) + ((uint32)(u16AdcReading
* 2344) >> 1)))  / 1000;

    vPrintf("u16AdcReading: %d \n", u16AdcReading);

    vPrintf("Battery Voltage (mV): %d \n", u16BattLevelmV);

    vAHI_AdcStartSample();
```

程序等待 ADC 转换完成之后， 读取了 ADC 的值， 并将读取值转化为电压值， 并通过串口输出。

接着进行下一次的采样。

可以通过超级终端 （设置 115200,8-n-1 无流控）连接测试板， 观察通过串口读取的电压值。

EEPROM

JN516x 带有 4Kbytes 的 EEPROM， 可以作为非易失性存储器使用。

在介绍如何使用 JN516x 的 EEPROM 部件之前， 需要说明：

不推荐开发者使用 EEPROM， 或者开发者应该非常谨慎的使用 EEPROM, 原因如下:

1. JenNet-IP 或者 ZigBee 节点通常使用 JenOS 的 Persistent Data Manager(PDM) 来访问 EEPROM, NXP 的 JenNet-IP 和 ZigBee 协议栈也提供了 PDM。 关于 JenOS 的使用说明请查阅 JN-UG-3075. 使用 PDM 可以带来的好处有：采用包装后的接口访问 EEPROM， 可以使用 ID 而不是地址来访问存储器。 如果同时使用 EEPROM 的 API 和 PDM 来访问 EEPROM， 将很容易造成访问的冲突。
2. ZigBee RF4CE 也会使用 EEPROM， 所以如果使用 ZigBee RF4CE 进行开发也必须避免访问的冲突。

初始化 EEPROM

u16AHI_InitialiseEEP()函数用来初始化 EEPROM.

JN516x 带有 4Kbytes 的 EEPROM, 被按照不同的分段进行组织。 初始化函数调用之后会返回两个值。 分别是：

1. 分段的数量
2. 每一个分段的字节数

EEPROM 的分段从 0 开始计数。

写入数据到 EEPROM

iAHI_WriteDataIntoEEPROMsegment() 函数用来将数据写入 EEPROM. 这一函数有四个参数：

u16SegmentIndex： 指定需要写入的分段索引号

u8SegmentByteAddress： 需要写入的分段内地址偏移

pu8DataBuffer：需要写入的 数据指针

u8Datalength： 数据长度

数据可以通过指定索引号写入任何一个 EEPROM 的分段， 而且可以通过指定分段内的地址偏移从任何位置开始写入。 但是需要注意， 这一函数不容许写入数据的长度溢出分段，也就是说如果地址偏移加上数据的长度如果超出了分段的最大容量， 那么写入操作将失败。

读取数据从 EEPROM

iAHI_ReadDataFromEEPROMsegment() 这一函数用来从 EEPROM 读取数据。 这一函数的参数和 iAHI_WriteDataIntoEEPROMsegment() 是一样的。 这里不再一一解释。 同样， 读取操作也不容许超出分段的边界。

擦除 EEPROM

iAHI_EraseEEPROMsegment() 函数用来擦除指定分段的 EEPROM

范例程序

我们随本手册带有操作 EEPROM 的范例程序， 您可以作为参考。 解压 JN-DEMO-EEPROM.zip 文件， 并导入到 Eclipse 中。

AN1174_154_Coord.c 文件包含了对于 EEPROM 的操作演示。

vInitSystem(void) 函数的后部对 EEPROM 进行了初始化， 并且获取了 EEPROM 的分段数量和每个分段的大小

```
u16SegmentsNumber=u16AHI_InitialiseEEP( &u8SegmentDatalength );
```

vProcessIncomingHwEvent(AppQApiHwInd_s *psAHI_Ind) 函数通过串口中断读取了串口数据，并将串口接收的数据写入 EEPROM， 然后读取, 在通过串口输出到超级终端。

```
/* If data has been received */

                        if ((psAHI_Ind->u32ItemBitmap & 0x000000FF) ==
E_AHI_UART_INT_RXDATA)

                        {
```

```
                cCharIn = (uint8)((psAHI_Ind->u32ItemBitmap & 0x0000FF00) >>
8);


                if (0==iAHI_WriteDataIntoEEPROMsegment(1,0,&cCharIn, 1))
                {
                        vPrintf("Write Success\n");
                        vPrintf("Now try to read data from EEPROM \n");

                        uint8 EEPROMData;
        iAHI_ReadDataFromEEPROMsegment(1,0,&EEPROMData, 1);

                        vPrintf("Data From EEPROM=%c \n", EEPROMData);
                }
                else
                {
                        vPrintf("Write Failed \n");
                }
        }
```

将程序编译下载到测试板之后， 您可以通过超级终端（设置 115200, 8-n-1 无流控）来向 EEPROM 写入一个字附的数据， 并且观察从 EEPROM 读出的数据。

PulseCounter： 脉冲计数

JN516x 带有 2 路脉冲计数通道。 分别是 Pulse Counter 0 和 Pulse Counter 1. 每一路计数通道都可以对外部信号通过 DIO 针脚进行脉冲计数。

两路脉冲计数通道都是 16 位的计数通道， 默认的接收 DIO1 和 DIO8 的信号输入。

Pulse Counter 0 可以选择配置接收 DIO 4 的信号输入， Pulse Counter 1 可以选择配置接收 DIO5 的信号输入。 两个计数器也可以联合起来作为一个 32bit 计数器使用， 在这种情况下， 可以选择作为信号输入的 DIO.

Pules Counter 可以在任何供电模式下工作， 包括休眠模式。 可以计数高达 100kHz 的信号输入。 可以配置上升或下降沿触发计数。 每个 Pulse counter 可以配置一个参考值， 用于在计数值大于参考值时产生中断， 也可以用于唤醒设备。

消除抖动

可以通过使用 32kHz 的时钟对输入信号进行抖动消除， 来避免对错误的信号边缘进行计数。可以进行下列消除抖动的设置：

1. 100kHz， 不使用消除抖动
2. 3.7kHz. 使用 2 次采样
3. 2.2 kHz, 使用 4 次采样
4. 1.2kHz, 使用 8 次采样

使用 Pulse Counter

bAHI_PulseCounterConfigure() 可以用来进行 Pulse Counter 的配置。 该函数包括以下参数

u8Counter： 所使用的 Pulse Counter 是 0 通道还是 1 通道

bEdgeType： 上升沿还是下降沿触犯， 上升沿： 0 ， 下降沿： 1

u8Debounce： 消除抖动设置。 0： 不使用消抖， 1： 2 次采样， 2： 4 次采样， 3： 8 次采样

u8Combine： 是否联合两个通道作为 32bit 计数器

bIntEnable： 是否使用中断。

vAHI_PulseCounterSetLocation()函数可以用来将 PC0 的输入信号配置到 DIO4, 或者将 PC1 的输入信号配置到 DIO5.

bAHI_SetPulseCounterRef() 用来对 PC 进行参考计数的设置， 需要注意的是， 脉冲计数达到参考计数值之后会产生中断， 也可以用来唤醒设备， 但是计数并不会停止， 脉冲计数会继续， 当达到最大值之后会重新复位。

bAHI_StartPulseCounter() 函数用来开始脉冲计数的计数。

bAHI_StopPulseCounter() 函数用来停止脉冲计数的计数。

bAHI_Clear16BitPulseCounter() ， bAHI_Clear32BitPulseCounter() 用来对计数器进行复位。调用之后计数值将被复位为 0。

bAHI_Read16BitCounter(), bAHI_Read32BitCounter() 用来读取计数器的计数值。

范例程序：

我们随本手册提供了脉冲计数的范例程序， 您可以参考。 解压 JN-Demo-PulseCounter.zip，并导入到 Eclipse 中。AN1174_154_Coord.c 代码中给出了使用脉冲计数的例子：

vInitSystem(void) 的后半部分对脉冲计数进行了初始化。

```
bAHI_PulseCounterConfigure(E_AHI_PC_1,
                           1,                    /* 下降沿触发          */
                           1,                    /* 2 次采样 消抖       */
                           E_AHI_PC_COMBINE_OFF,  /* 不使用联合计数*/
                           FALSE);               /* 不使用中断          */
    bAHI_StartPulseCounter(E_AHI_PC_1);
    bAHI_Clear16BitPulseCounter(E_AHI_PC_1);
```

程序使用 PC1 进行演示， 我们可以通过测试板上的拨码开关 1 来模拟脉冲输入。

vTickTimerISR 用来周期性的读取 PC1 的计数值并通过串口输出到超级终端 （115200, 8-n-1 无流控）。

```
uint16 u16Counter1;
bAHI_Read16BitCounter(E_AHI_PC_1, &u16Counter1);
vPrintf("PulseCounter = %d \n" , u16Counter1);
```

I2C 接口 - SHT 温湿度传感器

JN516x 可以使用 DIO17 和 DIO12 来实现 I2C 接口，并提供了相应的驱动函数来和外部传感器进行通讯。 DIO17 为 CLK, DIO12 为 SDA。

下面我们以连接 SHT 温湿度传感器为例来说明相应驱动的使用方法。

我们提供了相应的范例程序来演示 SHT 温湿度传感器的读取方法， 开发者可以解压 JN-Demo-SHT.zip 并导入 Eclipse 来查看相应的代码。

AN1174_154_Coord.c 文件中：

首先需要在 头部引入 驱动函数库的头文件。 这一库函数是由 Jennic 在开发包的平台函数中附带的。

```
#include "HtsDriver.h"
```

然后我们定义了相应的功能函数和变量来读取和保存温湿度数据。

```
PRIVATE void   vReadSHTSensors(void);

PRIVATE uint8  u8FindMin(uint8 u8Val1, uint8 u8Val2);


PRIVATE uint8 u8TempResult;

PRIVATE uint8 u8HtsResult;
```

vReadSHTSensors 函数调用了 HtsDriver.h 中的库函数实现了温湿度数据的读取。

```
PRIVATE void vReadSHTSensors(void)

{
```

```
    /* Read temperature, 0-52 are acceptable. Polls until result received */
    vHTSstartReadTemp();
    u8TempResult = u8FindMin((uint8)u16HTSreadTempResult(), 52);
    /* Read humidity, 0-104 are acceptable. Polls until result received */
    vHTSstartReadHumidity();
    u8HtsResult = u8FindMin((uint8)u16HTSreadHumidityResult(), 104);
}
```

在 vInitSystem(void) 过程中首先需要初始化传感器接口

```
vHTSreset();
```

然后我们在 Tick Timer 的回调函数中周期的读取温湿度数据并将数据通过超级终端显示

```
vReadSHTSensors();
vPrintf("\n\r Humidity = %d%%, Temperature = %d \n\r",u8HtsResult,u8TempResult);
```

您可以通过超级终端（设置参数 115200, 8-n-1 无流控）来查看 SHT 传感器所读取的数据

博讯所提供的测试板已经预留了 SHT 传感器的位置， 请查看 2.1 小节进行参考。

SPI

JN516x 通过可配置 IO 提供了 SPI 接口， 可以作为 SPI Master 或 Slave， 而且提供了封装好的 API 函数方便开发者对 SPI 接口进行操作。

这一小节我们将讲解如何使用 JN516x 作为 SPI Master 来操作 SPI Flash 芯片 M25P10.

需要说明的是， JN516x 芯片在启动时是默认关闭 SPI Master 的， 这一点和 JN514x 不一样，JN514x 默认打开 SPI Master 是为了可以从外部 Flash 启动。

SPI Master 使用如下 DIO 和 DO 作为接口

SPICLK: DO0

SPIMISO: DO1

SPIMOSI: DIO18

SPISEL0: DIO19

SPISEL1: DIO0 (可配置到 DIO14)

SPISEL2: DIO1 (可配置到 DIO15)

数据传输:

数据传输采用全双工模式， 所以数据可以同时输入和输出。 需要传输的数据会被缓存在 FIFO。 数据的大小可以选择从 1 到 32 位。 数据传输的顺序可以配置为 LSB 或者 MSB。

由于数据传输是同步的， 所以发送和接收数据使用 SPI Master 提供的同一个时钟，SPI 使用外围时钟， 最高可达 16Mbps 速率。

SPI Master API 介绍：

实现数据传输：

vAHI_SpiConfigure(): 在使用 SPI 接口之前， 需要使用这一函数对 SPI 接口参数进行配置。需要配置的参数如下：

SPI Slave 的数量

时钟频率

数据传输顺序， LSB 或者是 MSB

时钟极性

上升沿锁存数据 或 下降沿锁存数据

是否使用自动 Slave 选择

是否使用中断回调函数

vAHI_SpiSelect()： 这一函数用来选择 Slave， 如果在 SPI 初始化的时候关闭了自动 Slave 选择， 那么就需要通过这一函数手动选择 Slave。 如果传入参数 0 将取消对上一次 Slave 的选择。

vAHI_SpiStartTransfer()： 这一函数用来开始数据传输。

vAHI_SpiWaitBusy()： 这一函数将在 SPI 数据传输完毕的时候返回。 通常用来等待 SPI 数据传输完成，然后进行下一步操作。

u32AHI_SpiReadTransfer32(),u16AHI_SpiReadTransfer16(), u8AHI_SpiReadTransfer8()：这三个函数用来从 SPI 总线读取相应长度的数据。

以上函数就是我们常用的 SPI API 函数， 当然还有一些其他操作 SPI 的函数， 可以参考手册了解。

范例程序：

我们随本手册附带了一个演示如何利用 SPI API 来操作 SPI Flash 芯片来实现使用外部存储器的应用。 这里我们选用的是 M25P10 Flash 芯片来进行演示。 这一芯片具有 1Mbytes 的存储空间， 4 个 sector. 您可以解压 JN-DEMO-SPI.zip 文件并导入 Eclipse， 来查看范例的源代码。

首先我们需要了解一些关于 M25P10 的一些参数。 首先这一芯片具有 1Mbytes 的存储空间，分为 4 个 sector， 支持页编程的方式写入数据， 每个 Sector 包含 256 个 pages, 每个 page 是 128 bytes。 支持整体擦除和 sector 擦除。 支持硬件写保护和软件写保护。 支持低功耗模式。

下面介绍一下 M25P10 支持的指令集：

```
SPI_FLASH_WREN          0x06          /*  允许对 Flash 进行写操作      */
SPI_FLASH_WRDI          0x04          /*  禁止对 Flash 进行写操作      */
SPI_FLASH_RDSR          0x05          /*  读器件的状态寄存器           */
SPI_FLASH_WRSR          0x01          /*  写器件的状态寄存器           */
SPI_FLASH_READ          0x03          /*  读操作                       */
SPI_FLASH_BE            0xC7          /*  块擦除                       */
SPI_FLASH_SE            0xD8          /*  扇区擦除                     */
SPI_FLASH_PP            0x02          /*  页编程的方式写 Flash         */
SPI_FLASH_DP            0xB9          /*  低功耗模式                   */
SPI_FLASH_RES           0xAB          /*  低功耗模式恢复 , 读 electronic ID  */
```

程序中我们就将使用这些指令对芯片进行操作。

然后需要介绍一下 Status Register 状态寄存器。 状态寄存器可以用来了解芯片的工作状态或者设置芯片的工作模式。 具体的可以参考 M25P10 的手册。

通过 SPI_FLASH_RDSR 指令， 我们可以读取芯片的状态寄存器的值。 在范例程序中我们将使用 Bit 0 的 WIP（Write In Progress）标识了解芯片是否处于写工作状态。

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SRWD	0	0	0	BP1	BP0	WEL	WIP

打开范例工程的 AN1174_154_Coord.c 文件。

首先在程序的后部我们定义了一系列操作 SPI 的函数。 参考程序的注释我们就可以了解每个函数的用途。

vInitSystem 中我们调用 vSpiInit 进行了 SPI 接口的初始化。

然后 vProcessIncomingHwEvent 中我们对 SPI 进行了测试。 用户可以通过串口连接超级终端（设置： 115200-8-n-1, 无流控）。 然后在超级终端中输入任意字符就可以看到 SPI 测试的过程。

程序先设置了 M25P10 芯片的状态寄存器， 然后读取状态寄存器的值并显示。

然后程序对整个 Flash 进行了擦除。

程序定义了两个数组 u8Buff, u8ReadBuff。 分别用来保存需要写入的数据和读出的数据。数组的长度就是 Flash 芯片的 page 长度。

接着程序将数据写入 Flash， 然后把写入的数据读出并显示到超级终端。

6：其他参考手册向导

因为网站关于 JN516x 的文档非常多， 所以在这一章对 JN516x 的各种文档资料做一个索引，主要介绍各个文档的主要内容， 便于开发者根据问题的需要进行查阅。 这里所列举的文档并不是全部的， 我们只挑选了认为重要的文档进行说明。 更详细的介绍请参考北京博讯科技有限公司官方网站。

Datasheets

JN-DS-JN516x ：JN516x 芯片 Datasheet

http://www.boccn.com.cn/getfile.aspx?id=38

JN516x 的芯片 Datasheet, 介绍关于芯片的各种参数， 各种接口和针脚的定义。

JN-DS-JN5168-001-MO : JN5168 模块 Datasheet

http://www.boccn.com.cn/getfile.aspx?id=141

基于 JN516x 芯片的 JN5168 模块的 Datasheet. 针脚定义， 附录部分包含模块的 PCB footprint 以及天线辐射图. 可以作为用户基于模块开发产品的设计参考。

用户指南

JN-UG-3007 : Flash Programmer 使用说明

http://www.boccn.com.cn/getfile.aspx?id=249

下载工具 Flash Programmer 的使用说明。 如果用户对于如何下载程序有问题可以查阅这一文档。 附录包含如何安装 USB 转串口的驱动程序安装说明。

JN-UG-3064： 开发工具安装及配置

http://www.boccn.com.cn/getfile.aspx?id=250

开发平台的安装和使用说明。 介绍如何安装 JN516x 的开发工具集合， 以及如何配置开发工具。 如何安装各种协议栈 SDK. 关于 JN514x 的 SDK 如何安装也做了介绍。 内容还简单介绍了如何使用 Eclipse 开发环境进行初步的工程导入， 创建， 编译。

JN-UG-3087 ： 外围部件使用手册

http://www.boccn.com.cn/getfile.aspx?id=251

JN516x 外围接口部件的使用说明。 主要内容包括

1. 系统控制，电源管理， 供电监控
2. ADC
3. DIO
4. 通用串口 UART
5. 计时器 Timer
6. 唤醒 Timer
7. Tick Timer
8. Watchdog Timer
9. 脉冲计数器
10. SI 接口
11. SPI 接口
12. 扩展 Flash 存储的使用

文档的第二部分包括所有 API 的函数说明。 文档的第三部分包括各种中断回调函数的注册 API 以及硬件中断 Mask 码列表.

JN-UG-3024 ： IEEE802.15.4 协议栈开发手册

http://www.boccn.com.cn/getfile.aspx?id=1040

IEEE802.15.4 协议栈用户手册。 详细介绍了基于 802.15.4 协议栈的开发。 包含

1. 802.15.4 协议栈基础概念介绍
2. 协议栈 API
3. 基于项目模板进行应用开发
4. 802.15.4 协议栈 API 索引

JN-UG-3075： JenOs 用户指南

http://www.boccn.com.cn/getfile.aspx?id=270

JenOS 的用户指南。 包含 PDM（Persistent Data Manager）的使用说明

JN-UG-3080： JenNet-IP WPAN 协议栈开发手册

http://www.boccn.com.cn/getfile.aspx?id=271

JenNet-IP WPAN 的协议栈用户指南。 介绍了 JenNet-IP 无线网络协议的各种概念以及大致的 API 参考。 如果用户需要基于 JenNet-IP 协议构建无线网络， 需要参考这一手册。

JN-UG-3086： JenNet-IP LAN/WAN 协议栈开发手册

http://www.boccn.com.cn/getfile.aspx?id=273

JenNet-IP LAN/WAN 协议栈用户指南。 如何开发者需要构建 JenNet-IP 无线网络和 LAN 网络互通的应用， 需要参考这一手册。

参考手册

JN-RM-2027 ： 生产测试 API 手册

http://www.boccn.com.cn/getfile.aspx?id=254

生产测试 API 参考手册， 介绍了用于在产品设计和开发阶段进行产品测试的各种 API。

参考设计

JN-RD-6021： USB 编程电缆参考设计

http://www.boccn.com.cn/getfile.aspx?id=255

USB 编程电缆的参考设计。

JN-RD-6038： JN516x 模块参考设计

http://www.boccn.com.cn/getfile.aspx?id=257

JN516x 模块的参考设计。

应用说明

JN-AN-1001 ： 功耗计算说明

http://www.boccn.com.cn/getfile.aspx?id=1008

介绍了基于 802.15.4 的应用如何计算 JN516x 的功耗。 用户可以根据这一文档的计算进行电源的设计和选用。

JN-AN-1003 ： JN51xx boot loader 功能说明

http://www.boccn.com.cn/getfile.aspx?id=259

JN51xx boot loader 的功能说明

JN-AN-1035 ： 数据传输速率计算方法说明

http://www.boccn.com.cn/getfile.aspx?id=1009

介绍了如何计算基于 802.15.4 的应用如何计算数据传输速度的方法。

JN-AN-1059 ： 无线网络部署说明

http://www.boccn.com.cn/getfile.aspx?id=1010

介绍了如何部署无线网络的一些技术要点。 包括 如何选择天线， 如何部署无线节点的一些说明。

JN-AN-1174： 802.15.4 模板工程

http://www.boccn.com.cn/getfile.aspx?id=1036

802.15.4 的应用开发模板工程。 任何基于 802.15.4 协议栈的开发都需要基于这个工程模板开始。

JN-AN-1178： 802.15.4 无线串口应用

http://www.boccn.com.cn/getfile.aspx?id=1037

基于 802.15.4 协议栈的一个无线串口应用。 包含了实现代码和说明。 用户可以参考这一例程学习基于 802.15.4 协议栈的开发， 并且可以参考其代码学习串口的使用， 以及数据的传输。

JN-AN-1180： 802.15.4 家庭传感器网络应用

http://www.boccn.com.cn/getfile.aspx?id=1038

基于 802.15.4 协议栈的家庭传感器网络的应用。 包含代码和说明

JN-AN-1110： JenNet-IP WPAN/LAN 网关设备

http://www.boccn.com.cn/getfile.aspx?id=1046

开发 JenNet-IP 的 WPAN/LAN 网关设备的参考应用。

JN-AN-1162 ： JenNet-IP 智能家庭应用

http://www.boccn.com.cn/getfile.aspx?id=267

基于 JenNet-IP 智能家庭的演示应用。 包含代码和说明

JN-AN-1190： JenNet-IP 模板工程

http://www.boccn.com.cn/getfile.aspx?id=269

基于 JenNet-IP 进行开发的模板工程。 不推荐使用。 因为这一模板是基于 JN-AN-1162 的智能家庭应用的代码简化来的， 包含了很多冗余的代码， 结构也不够清晰。 建议开发者联系北京博讯科技有限公司索取更加容易使用的 JenNet-IP 模板工程。

BOCCN 博讯科技 北京博讯科技有限公司
Beijing Boccn Technology Co.,Ltd.

电话：**010-51663110**
传真：**010-51581150**

电子邮箱：**support@bocon.com.cn**
公司官网：**www.boccn.com.cn**