JN516x 集成外设 API 用户手册

广州周立功单片机科技有限公司

地址：广州市天河北路 689 号光大银行大厦 12 楼 F4

网址：http://www.zlgmcu.com

销售与服务网络

广州周立功单片机科技有限公司

地址：广州市天河北路 689 号光大银行大厦 12 楼 F4 邮编：510630

电话：(020)38730972 38730976 38730916 38730917 38730977

传真：(020)38730925

网址：http://www.zlgmcu.com

新浪微博：ZLG-周立功（http://weibo.com/ligongzhou）

广州专卖店

地址：广州市天河区新赛格电子城 203-204 室

电话：(020)87578634 87569917

传真：(020)87578842

南京周立功

地址：南京市珠江路 280 号珠江大厦 1501 室

电话：(025) 68123901 68123902

传真：(025) 68123900

北京周立功

地址：北京市海淀区知春路 113 号银网中心 A 座 1207-1208 室（中发电子市场斜对面）

电话：(010)62536178 62536179 82628073

传真：(010)82614433

重庆周立功

地址：重庆市石桥铺科园一路二号大西洋国际大厦（赛格电子市场）1611 室

电话：(023)68796438 68796439

传真：(023)68796439

杭州周立功

地址：杭州市天目山路 217 号江南电子大厦 502 室

电话：(0571)89719480 89719481 89719482
89719483 89719448 89719485

传真：(0571) 89719494

成都周立功

地址：成都市一环路南二段 1 号数码科技大厦 403 室

电话：(028) 85439836 85437446

传真：(028) 85437896

深圳周立功

地址：深圳市福田区深南中路 2072 号电子大厦 12 楼

电话：(0755)83781788（5 线）

传真：(0755)83793285

武汉周立功

地址：武汉市洪山区广埠屯珞瑜路 158 号 12128 室（华中电脑数码市场）

电话：(027)87168497 87168297 87168397

传真：(027)87163755

上海周立功

地址：上海市北京东路 668 号科技京城东座 7E 室

电话：(021)53083452 53083453 53083496

传真：(021)53083491

西安办事处

地址：西安市长安北路 54 号太平洋大厦 1201 室

电话：(029)87881296 83063000 87881295

传真：(029)87880865

厦门办事处

E-mail：sales.xiamen@zlgmcu.com

沈阳办事处

E-mail：sales.shenyang@zlgmcu.com

目　录

关于本文档

本手册描述了 JN516x 集成外设应用程序接口（API）和一个 NXP JN516x 系列无线微控制器外围功能的结合使用，阐述了每个外设的基本操作，并指出如何使用相应的 API 函数来控制 JN516x 器件上运行的应用程序中的外围功能。还对 C 函数以及 API 的相关资源进行了详细描述。

结构

本文档划分为 3 个部分：

- Part I:概念和操作信息包含 16 章：
  - Chapter 1 是 JN516x 集成外设 API 的功能概述；
  - Chapter 2 描述了包括 API 初始化函数在内的 API **通用函数**的使用；
  - Chapter 3 描述了**系统控制器函数**的使用，包括配置系统时钟和睡眠操作的函数在内；
  - Chapter 4 描述了使用**模拟外围函数**来控制 ADC 和比较器；
  - Chapter 5 描述了使用 **DIO 函数**来控制通用数字输入/输出引脚；
  - Chapter 6 描述了使用 **UART 函数**来控制兼容 16550 的 UART；
  - Chapter 7 描述了使用**定时器函数**来控制通用定时器；
  - Chapter 8 描述了使用**唤醒定时器函数**来控制可以计时器睡眠时间的唤醒定时器；
  - Chapter 9 描述了使用**节拍定时器函数**来控制高精度硬件定时器；
  - Chapter 10 描述了使用**看门狗定时器函数**来控制看门狗，看门狗允许避开软件锁定；
  - Chapter 11 描述了使用**脉冲计数器函数**来控制 2 个脉冲计数器；
  - Chapter 12 描述了使用**红外发送器函数**来控制定时器 2 的红外发送功能；
  - Chapter 13 描述了使用**串行接口（SI）函数**来控制一个 2 线 SI 主机和 SI 从机；
  - Chapter 14 描述了使用**串行外设接口（SPI）主机函数**来控制 SPI 总线的主机接口；
  - Chapter 15 描述了使用**串行外设接口（SPI）从机函数**来控制 SPI 总线的从机接口；
  - Chapter 16 描述了使用 **Flash 存储器函数**来管理 Flash 存储器；
- Part II: 参考信息包含 15 章：
  - Chapter 17 详细描述了 API 的**通用函数**，包括 API 初始化函数在内；
  - Chapter 18 详细描述了**系统控制器函数**，包括配置系统时钟和睡眠操作的函数在内；
  - Chapter 19 详细描述了用来控制 ADC 和比较器的**模拟外设函数**；
  - Chapter 20 详细描述了用来控制通用数字输入/输出引脚的 **DIO 函数**；
  - Chapter 21 详细描述了用来控制兼容 16550 UART 的 **UART 函数**；
  - Chapter 22 详细描述了用来控制通用定时器的**定时器函数**；
  - Chapter 23 详细描述了用来控制唤醒定时器的**唤醒定时器函数**，唤醒定时器用来计时睡眠时间；
  - Chapter 24 详细描述了用来控制高精度硬件定时器的**节拍定时器函数**；
  - Chapter 25 详细描述了用来控制看门狗的**看门狗定时器函数**，看门狗允许避开软件锁定；

- Chapter 26 详细描述了用来控制 2 个脉冲计数器的**脉冲计数器函数**；
- Chapter 27 详细描述了用来控制红外发送的**红外发送器函数**；
- Chapter 28 详细描述了用来控制一个 2 线 SI 主机和 SI 从机的**串行接口（SI）函数**；
- Chapter 29 详细描述了用来控制 SPI 总线主机接口的**串行外设接口(SPI)主机函数**；
- Chapter 30 详细描述了用来控制 SPI 总线从机接口的**串行外设接口(SPI)从机函数**；
- Chapter 31 详细描述了用来管理 Flash 存储器的 **Flash 存储器函数**；

● Part III: 附录描述了处理外围设备中断的相关信息。

规约

文件、文件夹、函数和参数类型**加粗**表示。

函数参数用*斜体*表示。

代码段用 Courier New 字体表示。

 这是 Tip（提示），它表示有用或实用的信息。

 这是 Note（注），它强调其它重要的信息。

 这是 Caution（警告），它警告可能会导致设备故障或损坏的情况。

缩写词

ADC	模数转换器
AES	高级加密标准
AHI	应用硬件接口
API	应用程序接口
CPU	中央处理单元
CTS	清除发送
DAC	数模转换器
DAI	数字音频接口
DIO	数字输入/输出
EIRP	等效全向辐射功率
FIFO	先入先出（队列）
GPIO	通用输入/输出
LPRF	低功耗射频
MAC	媒体访问控制
NVM	非易失性存储器
PWM	脉宽调制
RAM	随机存取存储器
RTS	准备好发送
SI	串行接口
SPI	串行外设接口

UART	通用异步收发器
VBO	电压掉电

相关文档

JN-DS-JN516x	JN516x 数据手册

商标

所有商标的所有权属于它们的所有者。

Part I 概念和操作信息

第1章 简介

本章介绍 JN516x 集成外设应用程序接口（API），它和一个 NXP JN516x 系列无线微控制器的外围功能结合起来使用。JN516x 系列器件的外设相同，但存储器大小不同：

- JN5168（32KB RAM，4KB EEPROM，256KB Flash 存储器）；
- JN5164（32KB RAM，4KB EEPROM，160KB Flash 存储器）；
- JN5161（8KB RAM，4KB EEPROM，64KB Flash 存储器）。

1.1 JN516x集成外设

JN516x 微控制器带有大量片内外设，可供微控制器 CPU 运行的用户应用程序使用。这些‘集成外设’包括：

- 系统控制器
- 模拟外设：
  - 模数转换器（ADC）
  - 比较器
- 数字输入/输出（DIO）
- 通用异步收发器（UART）
- 定时器
- 唤醒定时器
- 节拍定时器
- 看门狗定时器
- 脉冲计数器
- 串行接口（2 线）：
  - SI 主机
  - SI 从机
- 串行外设接口（SPI）：
  - SPI 主机
  - SPI 从机
- 外部 Flash 存储器接口

上述外设请见图 1.1。

关于这些外设的硬件详细信息，请参考相应的芯片数据手册。

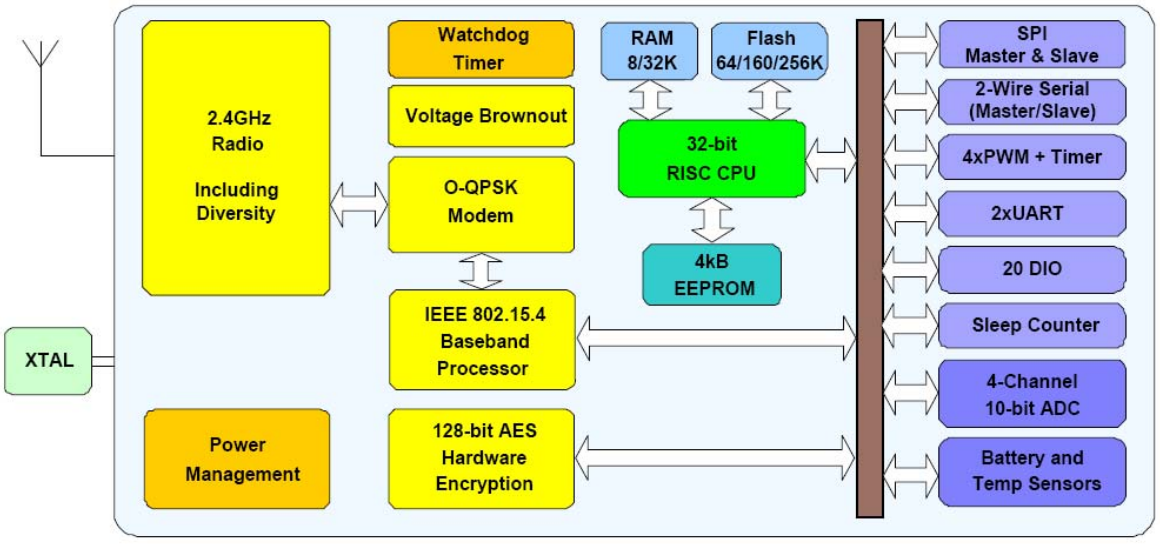

图 1.1 JN516x 功能框图

1.2 JN516x集成外设API

JN516x 集成外设 API 是一个 C 函数集合，JN516x 无线微控制器上运行的应用程序代码可以使用它来控制第 1.1 节中列出的片内外设。这个 API（有时称为 AHI）在头文件 **AppHardwareApi.h** 中定义，头文件包含在 JN516x 器件的 NXP 软件开发者套件（SDK）中。API 调用的软件位于片内 ROM 中。

这个 API 在片内寄存器上方提供了一个薄的软件层，用来控制集成的外设。API 将几个寄存器访问打包到一个函数调用中来简化外设的使用，不再需要详细了解外设的操作情况。

警告:

JN516x 集成外设 API 函数是不可重入的。必须允许函数先执行完再被重新调用，否则可能出现不希望的结果。

请注意，集成外设 API 不包括控制下列功能部件的函数：

- 内置到 JN516x 器件的 IEEE 802.15.4 基带处理器—— 这由无线网络协议栈软件（可能是一个 IEEE 802.15.4、ZigBee、JenNet 或 JenNet-IP 栈）来控制，为此给 API 提供了合适的栈软件产品；
- 内置到 JN516x 器件的 128 位 AES 硬件加密内核—— 这使用 *AES 协处理器 API 参考手册（JN-RM-2013）*中描述的函数来控制；
- EEPROM—— 这使用 Jennic 操作系统（JenOS）中的持久数据管理器来控制。更深入的详细情况请参考 *JenOS 用户指南（JN-UG-3075）*;
- JN516x 评估套件电路板的资源，例如，传感器和显示面板（尽管评估套件电路板上的按钮和 LED 连接到 JN516x 器件的 DIO 引脚）—— 一个称为 LPRF 板 API 的特殊函数库，由 NXP 专门提供，具体描述请参考 *LPRF 板 API 参考手册（JN-RM-2003）*。

1.3 如何使用本手册

本手册的剩余内容基本按照每个外设模块一章的方式来组织。用户应当按照下面的描述来使用本手册：

1）首先学习第 2 章，这一章描述的是通用函数，不是与某个特定特定外设模块相关的特定函数。该章阐述了如何初始化集成外设 API，以便在用户的应用程序代码中使用。

2）接下来学习第 3 章，该章描述了系统控制器相关的特性。用户可能需要在应用中使用到这些特性。

3）然后学习 Part I：概念和操作信息中的章节，它们和用户希望在应用中用到的特定外设相对应。

关于引用的 API 函数的详细情况，请参考 Part II：参考信息。关于中断处理的描述请参考 Part III：附录。

第2章 通用函数

本章描述了‘通用函数’的使用，这些函数与任何外设模块都无关，但用户的应用程序代码可能会需要它们，其中的 API 初始化函数是一定要用到的。

这些函数执行的功能包括：

- API 初始化；
- 无线发送功率的配置；
- 随机数发生器的使用；
- 访问 JN516x 内部非易失性存储器。

2.1 API初始化

在调用 JN516x 集成外设 API 的任何其他函数之前，必须先调用 **u32AHI_Init( )**来初始化 API。每次 JN516x 微控制器复位和睡眠唤醒之后都必须调用这个函数。

警告:

如果用户正在使用 JenOS（Jennic 操作系统），不得在用户代码中明确调用 **u32AHI_Init( )**，因为这个函数由 JenOS 在内部调用。这主要供正在开发 ZigBee PRO 应用的用户使用。

2.2 无线发送功率

一个 JN516x 器件的无线发送功率可以是变化的。一个标准 JN516x 模块的发送功率范围是-32~+2.5dBm。

发送功率可以使用 **bAHI_PhyRadioSetPower( )**来设置。这个函数允许用户将功率设置成功率范围内 4 个可能的级别，关于这些级别的详细信息请参考第 17 章的功能描述。

注意，只有在调用完 **vAHI_ProtocolPower( )**来使能协议功率域之后（见第 3.2.1 节）才调用 **bAHI_PhyRadioSetPower( )**。

2.3 天线分集

JN516x 器件提供一个天线分集装置，允许 2 根天线连接到器件上。如果这个特性被采纳，并且确认当前天线的发送和/或接收性能比较差，就自动切换到备用天线。

为了使用天线分集，2 根天线必须通过一个 2-state开关连接到JN516x器件，器件使用引脚 DIO12 和DIO13 的一对互补信号输出对开关进行控制。在一个位置（例如，DIO12-13 = 10），开关将JN516x器件的RF_IN引脚与一根天线相连；在另一个位置（例如，DIO12-13 = 01），开关将这个引脚与另一根天线相连。该连接如下面的图 2.1所示。

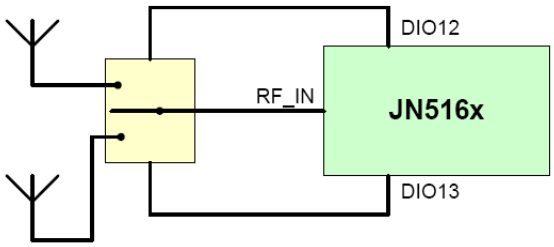

图 2.1 天线分集的连接

DIO12 和 DIO13 引脚必须首先通过调用 **vAHI_AntennaDiversityOutputEnable( )**函数来使能，使其可以用于天线分集。

天线分集在应用程序中通过调用函数 **vAHI_AntennaDiversityEnable( )**来使能。这个函数允许天线分集单独使能用于发送和接收通路（或用于两个通路）。发送和接收时天线分集的操作情况如下所述：

- **发送**：对于发送来说，是否切换天线取决于 IEEE 802.15.4 MAC 应答的使用。一旦发送一个 IEEE 802.15.4 包，无线收发器就进入接收模式并等待目标节点的应答。如果未接收到应答，器件就重新尝试在备用天线上发送（重试的次数可在 IEEE 802.15.4 MAC 中配置）。以后每次尝试时会切换到所选的天线；
- **接收**：对于接收，JN516x 器件每隔 40μs 对相应无线通道中接收到的能量进行测量。测得的能量级别与预先设置的能量阈值相比较。如果测量结果低于这个阈值并且下面所有的条件都成立，JN516x 器件将自动切换天线：
  - 无线外设不处在接收包的过程中；
  - 在过去的 40μs 内尚未检测到信号质量高于指定最低阈值的前导符；
  - 无线外设不在等待前一次传输的应答。

信号能量和信号质量阈值在应用程序中可以使用函数 **vAHI_AntennaDiversityControl( )** 来设置。

当前的天线分集状态可以使用函数 **u8AHI_AntennaDiversityStatus( )**来获得。这个函数返回发送最后一个包所使用的天线，接收最后一个包所使用的天线以及当前所选的天线。

当前所选的天线可以通过调用函数 vAHI_AntennaDiversitySwitch( )来手动切换。通常不要求调用这个函数，因为希望大多数应用能够利用自动发送和/或接收天线分集控制特性，该特性通过调用 **vAHI_AntennaDiversityEnable( )**来使能。

2.4 随机数发生器

JN516x 器件带有一个随机数发生器，它可以在下面其中一种模式下产生 16 位的随机数：

- **单触发模式**：发生器产生一个随机数然后停止；
- **连续模式**：发生器连续运行，每隔 256μs 产生一个新的随机数。

随机数发生器可以使用函数 **vAHI_StartRandomNumberGenerator( )**在上述任何一种情况下启动。这个函数还允许在一个随机数变得可用时产生的中断被使能，这被当做一个系统控制器中断，由使用函数 **vAHI_SysCtrlRegisterCallback( )**注册的回调函数来处理（见第 3.5 节）。

此后随机产生的值可以使用函数 **u16AHI_ReadRandomNumber( )**来读取。一个新随机数的可用性以及是否需要调用‘读’函数可以使用下面其中一种方法来决定：

- 等待一个随机数发生器中断（如果使能），见上面的描述；
- 周期性地调用函数 **bAHI_RndNumPoll( )**来轮询一个新随机值的可用性。

当在连续模式下运行时，随机数发生器可以使用函数 **vAHI_StopRandomNumber Generator( )**来停止。

注：

随机数发生器使用 32kHz 的时钟域（见第 3.1 节），如果使用一个高精度的外部 32kHz 时钟源则无法正确运行。因此，如果用户在应用中要产生随机数，建议用户使用内部 RC 振荡器或一个低精度的外部时钟源。在应用中用户也可以先产生随机数再切换到一个高精度的外部时钟。

2.5 访问内部NVM

JN516x 器件包含一小块非易失性存储器（NVM），组成 4 个 32 位的字，编号为 0、1、2 和 3。这个存储器可以在 JN516x RAM 不工作时（例如，RAM 不保持的睡眠期间）用来保存重

要数据（例如，计数器值）。

有两个函数用来访问这个存储器：

- **vAHI_WriteNVData( )**：用来将一个 32 位字的数据写入 4 个存储器单元中的一个；
- **u32AHI_ReadNVData( )**：用来从其中一个存储器单元中读出一个 32 位字的数据。

警告:

微控制器完全断电时这个 JN516x NVM 的内容不保留。但是，器件复位时 NVM 的内容保持不变。

第3章 系统控制器

本章描述如何使用函数来控制系统控制器的特性。

这些函数执行的功能包括：

- 时钟管理；
- 电源管理；
- 电源电压监控；
- 芯片复位；
- 中断。

3.1 时钟管理

系统控制器给 JN516x 微控制器提供时钟，分成 4 个主要模块：系统时钟域、外设时钟域、CPU 时钟域和一个 32kHz 的时钟域。

系统时钟域

系统时钟是一个高速参考时钟，芯片正式工作时外围时钟和 CPU 时钟都来自于该时钟。这个域的时钟来源于下面其中一个时钟源：

- 外部 32MHz 晶体振荡器；
- 内部高速 RC 振荡器；

晶体振荡器由器件引脚 4 和 5 连接的 32MHz 外部晶体来驱动。在晶体振荡器的作用下，这个域产生一个 32MHz 的系统时钟。

未校准的 RC 振荡器通常在 27MHz 下运行，但经过校准后可以运行在近似 32MHz 下。由于 RC 振荡器比晶体振荡器启动快得多，因此主要用于睡眠后的全速启动。

当系统时钟由 RC 振荡器提供时，无线收发器和一些外设不应该使用。系统时钟的启动和时钟源的选择请见第 3.1.1 节和 3.1.2 节的描述。

外设时钟域

外设时钟来自于系统时钟，用作包括调制解调器和基带处理器在内的片内外设的时钟参考。外设时钟的工作频率是系统时钟频率的一半。当系统时钟由外部 32MHz 的晶体振荡器提供时，外设时钟运行在 16MHz。

CPU 时钟域

CPU 时钟由系统时钟分频得到，用作微处理器和存储器子系统的时钟参考。CPU 时钟频率选择请见第 3.1.3 节的描述。

32kHz 时钟域

32kHz 时钟域主要在低功耗睡眠期间使用（还供 JN516x 器件上的随机数发生器使用，见第 2.4 节）。当处于睡眠模式时（见第 3.2.3 节），CPU 不运行，依靠中断来唤醒。中断可以由一个片内唤醒定时器产生（见第 8 章），或者由一个外部源通过一个 DIO 引脚（见第 5 章）、一个片内比较器（见第 4.3 节）或一个片内脉冲计数器（见第 11 章）来产生。唤醒定时器由 32kHz 域来驱动。这个域的 32kHz 时钟可以来源于下面其中一个时钟源：

- 内部 RC 振荡器；
- 外部晶体；
- 外部时钟模块；

晶体振荡器由 DIO9 和 DIO10 连接的一个外部 32kHz 晶体来驱动。如果使用外部晶体，外

部时钟模块就连接到 DIO9。

这个域的源时钟选择见第 3.1.4 节的描述。

32kHz 域在芯片正常工作时仍然有效，可以以外设时钟为基准进行校准来提高计时精度，见第 8.2 节。

3.1.1 系统时钟启动和源选择

根据第 3.1 节的介绍，JN516x 器件的系统时钟可能有两个时钟源：

- 内部高速 RC 振荡器；
- 外部晶体振荡器；

其中，晶体振荡器提供的时钟比 RC 振荡器更精确。

复位之后，JN516x 器件从内部高速 RC 振荡器获取系统时钟。默认情况下，只要晶体振荡器稳定下来（这可能需要 1ms 的时间）就自动切换到外部 32MHz 晶体振荡器。应用程序代码在复位后立刻执行。

只要器件和系统时钟完全启动和运行，系统时钟源就可以使用函数 **vAHI_SelectClockSource( )**来更改。当前的源时钟可以通过调用函数 **bAHI_GetClkSource( )**来识别。

调用函数 **bAHI_TrimHighSpeedRCOsc( )**，RC 振荡器可以通过校对来提高频率精度。

使用 RC 振荡器时要重点注意以下限制：

- 如果未经过校对，RC 振荡器就产生一个±18%精度的系统时钟频率（经过校对后的频率精度为±5%）；
- 使用 RC 振荡器时整个系统不能运行。可能可以执行代码，但不能成功发送或接收无线信号。而且，外围时钟可能不够精确，无法支持某个外设功能，例如，UART 通信。

因此，使用 RC 振荡器时，不应该尝试使用无线收发器，使用 JN516x 外设时应该特别小心。

3.1.2 睡眠之后的系统时钟启动

默认情况下，在睡眠之后，JN516x 器件从内部高速 RC 振荡器获取系统时钟，但只要晶体振荡器稳定下来（可能需要 1ms 的时间）就自动切换成外部 32MHz 的晶体振荡器。这样，应用程序代码在睡眠之后立即执行。

如果未发生时钟自动切换，就可能继续使用内部高速 RC 振荡器。这时，在进入睡眠之前就必须调用函数 vAHI_EnableFastStartUp( )，选择手动切换选项，取消自动切换到晶体振荡器。

3.1.3 CPU时钟频率选择

JN516x 器件的 CPU 时钟频率是一个范围。默认情况下，源时钟频率减半来产生 CPU 时钟。因此：

- 使用外部晶体振荡器，32MHz 的源频率将产生一个 16MHz 的 CPU 时钟频率；
- 使用未经校准的内部高速 RC 振荡器，27MHz 的源频率将产生一个 13.5MHz（±18%）的 CPU 时钟频率；

注：

通过调用 **bAHI_TrimHighSpeedRCOsc( )**，高速 RC 振荡器的频率可以调节成一个已校准的 32MHz 频率

但是，备用的 CPU 时钟频率可以使用函数 bAHI_SetClockRate( )来配置。必须指定一个分频因子来分频源时钟，产生 CPU 时钟。可能的分频因子有 1、2、4、8、16 和 32：

- 对于一个 32MHz 的源时钟，可能的 CPU 时钟频率是 1、2、4、8、16 和 32MHz；

- 对于一个 27MHz 的源时钟，可能的 CPU 时钟频率是 0.84、1.17、3.38、6.75、13.5 和 27MHz。

3.1.4 32kHz时钟选择

根据第 3.1 节的介绍，JN516x 器件可以选择 32kHz 时钟的源时钟。下面就详细描述这个源时钟的选择。

注:

默认的时钟源是内部 32kHz RC 振荡器。如果需要一个外部 32kHz 的时钟源，就只要调用下面描述的函数。一旦已经选择了一个外部源，就不可能再切换回内部 RC 振荡器。

32kHz 时钟可以选择源自一个外部晶体或时钟模块。如果需要其中一个外部时钟源，就必须调用函数 **bAHI_Set32KhzClockMode( )**。如果需要，这个函数应该在靠近应用程序的起始处被调用。关于使用这个函数来选择一个外部晶体的更多信息详见下面的描述。

如果选择外部晶体振荡器，那么在应用程序使用定时器 0 或任何唤醒定时器之前必须调用 **bAHI_Set32KhzClockMode( )**，因为在将时钟源切换成外部晶体时函数要使用这些定时器。这个函数启动外部晶体，启动过程可能需要 1s 的时间稳定下来，函数等待晶体准备好再返回。

如果选择外部晶体振荡器，为了启动外部晶体，可以选择先调用函数 **vAHI_Init32KhzXtal( )**。这个函数立刻返回，允许应用程序执行其他处理，或在等待晶体稳定下来时允许应用程序使 JN516x 器件进入睡眠模式。在睡眠的情况下，应用程序通常应当设置一个唤醒定时器在 1s 后将器件唤醒。然后，为了将 32kHz 的时钟源切换成外部晶体，必须调用 **bAHI_Set32KhzClockMode( )**。

如果选择外部时钟模块（RC 电路），可以使用函数 **vAHI_Trim32KhzRC( )**设置电路的电流消耗来选择所产生时钟频率的精度。

外部时钟源必须按照如下方式连接：

- 外部时钟模块必须连接到 DIO9。用户必须首先使用函数 **vAHI_DioSetPullup( )**禁能 DIO9 的上拉；
- 外部晶体振荡器必须连接到 DIO9 和 DIO10。DIO9 和 DIO10 的上拉自动禁能。

请注意，没必要明确地将 DIO9 或 DIO10 配置成输入，因为这会通过 **bAHI_Set32KhzClockMode( )**或 **vAHI_Init32KhzXtal( )**自动处理。

3.2 电源管理

本节描述如何使用集成外设 API 来控制一个 JN516x 微控制器的功率，包括对某些片内外设供电的电源稳压器的控制以及低功耗睡眠模式管理的控制。

3.2.1 电源域

一个 JN516x 微控制器有许多电源域，如同下述：

- **数字逻辑域：**这个域给 CPU 和数字外设以及无线收发器（包括加密协处理器和基带控制器）供电。这个域到无线收发器的时钟可以由应用程序来使能/禁能（见第 3.2.2 节）。该域在睡眠期间一直不通电；
- **模拟域：**这个域给 ADC 供电。该域在调用函数 **vAHI_ApConfigure( )**来配置模拟外设时启用（见第 4 章）。这个域在睡眠期间一直不通电；
- **RAM 域：**这个域给片内 RAM 供电。该域在睡眠期间可能通电，也可能不通电；
- **无线域：**这个域给无线收发器供电。该域在睡眠期间一直不通电；

- **VDD 电源域：**这个域给唤醒定时器、DIO 模块、比较器和 32kHz 振荡器供电。该域由外部电源（电池）驱动，始终通电。但是，唤醒定时器和 32kHz 振荡器在睡眠期间可能通电，也可能不通电。

CPU（数字逻辑域）和片内 RAM 的独立电压稳压器可灵活实现不同的低功耗睡眠模式，允许存储器在 CPU 掉电时通电（存储器内容保持不变）或不通电，关于睡眠模式的更多信息请参考第 3.2.3 节。

3.2.2 无线收发器时钟

到无线收发器的时钟可以使用函数 **vAHI_ProtocolPower( )**来使能/禁能。但是，在复位或睡眠周期以外禁能这个时钟必须十分小心。应该注意以下几点：

- 禁能这个时钟是提供这个时钟但不使用（选通）；
- 禁能这个时钟会导致 IEEE 802.15.4 MAC 设置丢失。因此，在禁能这个时钟之前用户必须保存好当前的 MAC 设置。当重新使能时钟时，MAC 设置必须恢复成保存的设置。用户可以使用 802.15.4 栈 API 的函数来保存和恢复 MAC 设置，具体请参*考 IEEE 802.15.4 栈用户指南（JN-UG-3024）*的描述：
  - 使用函数 **vAppApiSaveMacSettings( )**来保存 MAC 设置；
  - 使用函数 **vAppApiRestoreMacSettings( )**来恢复保存的 MAC 设置。这个函数调用 **vAHI_ProtocolPower( )**后时钟自动重新使能。
- 切勿在 802.15.4 MAC 层激活时调用 **vAHI_ProtocolPower( )**来禁能时钟，否则微控制器可能冻结；
- 时钟禁能时不要对栈进行任何调用，因为这可能导致栈试图访问相关的硬件（硬件是禁能的），并因此而导致异常。

3.2.3 低功耗模式

为了在器件不需要完全激活期间节省功耗，JN516x 微控制器能够进入许多低功耗模式。通常有 2 种低功耗模式，睡眠模式（包括深度睡眠模式）和休眠模式，见下面的描述。

睡眠和深度睡眠模式

在睡眠模式下，为了节省功耗，大多数内部芯片功能被关断，包括 CPU 和大部分片内外设。但是，DIO 引脚的状态保留，包括输出值和上拉使能，这就使得任何与外部相连的接口都保留下来。在睡眠期间，片内 RAM、32kHz 振荡器、比较器和脉冲计数器可以选择保持有效。

当可以选择 4 种睡眠模式中的其中一种时，睡眠模式使用函数 **vAHI_Sleep( )**来启动，选择哪种睡眠模式取决于 RAM 和 32kH 振荡器是否关断。睡眠过程中 32kHz 振荡器和 RAM 是否有效概括如下：

- **32kHz 振荡器：**理论上，32kHz 振荡器（内部 RC、外部时钟或外部晶体）在睡眠期间要么留下来继续运行，要么停止运行。但是，这个振荡器由唤醒定时器使用，如果唤醒定时器将用来使器件从睡眠中唤醒，它就必须留下来继续运行。而且，如果这个振荡器使用一个外部源，就不建议在进入睡眠模式时终止振荡器；

  注：如果脉冲计数器将在器件睡眠时去抖运行，32kHz 振荡器就必须留下来继续运行，请参考第 11 章。

- **片内 RAM：**在睡眠期间，片内 RAM 可以维持供电或停止供电。微控制器完全运行时，应用程序、堆栈上下文数据和应用数据全部保存在片内 RAM 中，一旦 RAM 的电源被关断，这些数据就丢失。

- 如果睡眠期间 RAM 的电源被切断，退出睡眠模式时应用程序就从片内 Flash 存储器重装到 RAM 中，如果堆栈上下文和应用数据在进入睡眠模式之前被保存到片内 EEPROM 中，应用程序也可以将它们重装；
- 如果睡眠期间保持给 RAM 供电，应用程序和数据就保留。这对于短时间的睡眠很有用，与睡眠时间相比，唤醒将应用程序和数据重装到 RAM 所花费的时间更长。

选择深度睡眠模式可以更进一步地节省功耗，在这个模式下，CPU、RAM 以及系统和 32kH 时钟域都关断。除此之外，所有外部 Flash 存储器也关断。显而易见，深度睡眠模式比睡眠模式能节省更多的功耗。

注：正常睡眠模式期间外部 NVM 继续供电。如果需要，用户可以使用函数 **AHI_FlashPowerDown( )**来关断一个外部 Flash 存储器设备，如果用户正在使用一个兼容的 Flash 器件，**vAHI_FlashPowerDown( )**必须在 **vAHI_Sleep( )**之前调用。更多详细信息请参考 16.4 节。

可以通过下面其中一种方式将微控制器从睡眠模式唤醒：

- DIO 中断；
- 唤醒定时器中断（需要 32kHz 振荡器运行）；
- 比较器中断；
- 脉冲计数器中断。

器件只能通过将复位线拉低或通过一个触发 DIO 引脚改变的外部事件从深度睡眠模式唤醒。

当器件重启时，在冷启动或热启动入口点开始处理，由器件从中唤醒的睡眠模式决定。

休眠模式

休眠模式是一种低功耗模式，在这种模式下，CPU、RAM、无线收发器和数字外设保持供电，但是 CPU 的时钟停止运行（所有其他时钟正常运行）。休眠模式比睡眠模式节省的功耗要少，但能更快地恢复到全面运行模式。对于需要极短时间低功耗的情况，例如，等待一个定时器事件或等待一个传输完成，休眠模式很有用。

CPU 可以通过调用函数 **vAHI_CpuDoze( )**来进入休眠模式，之后通过任何一个中断退出休眠模式。

3.2.4 电源状态

JN516x 微控制器的电源状态使用函数 **u16AHI_PowerStatus( )**来获得。这个函数返回一个位图表，指示是否发生了以下情况：

- 器件已经结束一个睡眠-唤醒循环；
- 睡眠期间 RAM 内容保持不变；
- 模拟电源域接通；
- 协议逻辑可运行（时钟使能）；
- 看门狗超时导致上次器件重启；
- 32kHz 时钟准备好（例如，复位或唤醒之后准备好）；
- 器件已经退出深度睡眠模式（而不是复位）。

关于位图表的更多详细信息请参考第 18 章的函数描述。

3.3 电源电压监控器（SVM）

‘掉电’是指一个器件或系统的电源电压下降至低于一个预先定义的电平，这可能妨碍器

件/系统的操作或对器件/系统的操作有害。JN516x 微控制器有一个电源电压监控器（SVM）对掉电条件进行检测。SVM 可以通过集成外设 API 的函数来配置和监控。

3.3.1 配置SVM

默认情况下，JN516x 器件的 SVM 特性自动使能，掉电电压设为 2.0V。检测到掉电时芯片自动复位。

通过调用函数 **vAHI_BrownOutConfigure( )**，SVM 设置可以由默认值变为其他值，这个函数允许执行下列配置：

- **SVM 使能/禁能：**SVM 特性可以使能/禁能。如果配置函数被调用，并且需要 SVM，SMV 特性就必须在函数中明确使能；
- **掉电电平：**掉电电压电平可以设置成下面其中一个值：1.95V、2.0V（默认值）、2.1V、2.2V、2.3V、2.4V、2.7V 或 3.0V；
- **掉电时复位：**可以使能/禁能掉电自动复位；
- **掉电中断：**可以使能/禁能掉电相关的 2 个独立中断：
  - 器件进入掉电状态时（电源电压下降至低于掉电电压电平）产生一个中断；
  - 器件退出掉电状态时（电源电压上升至高于掉电电压电平）产生一个中断。

配置函数返回后，新的设置生效前有一段延迟，这个延时高达 3.3μs。

注:

器件复位或睡眠之后，重新设置 SVM 的默认配置。

3.3.2 监控电压

如果 SVM 使能（见第 3.3.1 节），JN516x 器件的掉电状态可以用下面 3 种方式中的其中一种来监控：自动复位、中断或轮询。下面对这些方式进行描述。

掉电时自动复位

掉电时自动复位默认使能，但也可以通过函数 **vAHI_BrownOutConfigure( )**来使能/禁能。芯片复位后，通过调用函数 **bAHI_BrownOutEventResetStatus( )**，应用可以检查掉电是否是产生复位的原因。

掉电中断

器件进入掉电状态和/或退出掉电状态时可以产生中断。这两个中断可以通过函数 **vAHI_BrownOutConfigure( )**单独使能/禁能。掉电中断是系统控制器中断，由使用函数 **vAHI_SysCtrlRegisterCallback( )**注册的回调函数来处理，见第 3.5 节。

掉电轮询

如果掉电中断和自动复位被禁能(但 SVM 仍然使能)，利用函数 **u32AHI_BrownOutPoll( )**，器件的掉电状态可以通过手动轮询获得。这个函数将指示电源电压当前高于还是低于掉电电平。

3.4 复位

JN516x 微控制器可以使用函数 **vAHI_SwReset( )**从应用中复位。这个函数启动芯片的整个复位序列，等效于将外部 RESETN 线拉低。注意，在芯片复位过程中，片内 RAM 的内容可能丢失。

RESETN 线也可能连接一个或多个外部器件。因此，和它相连的任何外部器件都可能受影响。

注:

为了产生复位，一个外部 RC 电路可以与 RESETN 线相连。需要的电阻和电容值请查看微控制器的数据手册。

3.5 系统控制器中断

系统控制器中断包括许多片内外设的中断，这些外设没有自己的中断：

- 比较器；
- DIO；
- 唤醒定时器；
- 脉冲计数器；
- 随机数发生器；
- 掉电检测器。

这些外设的中断可以使用自己集成外设 API 中对应的函数来单独使能。

这些中断源中断的处理必须包含在一个用户定义的回调函数中，回调函数使用函数 **vAHI_SysCtrlRegisterCallback( )**来注册。当一个类型 E_AHI_DEVICE_SYSCTRL 的中断出现时，自动调用已注册的回调函数。然后，确切的中断源（在上面列举的外设中）可以从传递到函数中的位图表来识别。请注意，中断将在回调函数调用之前自动清除。

注:

回调函数的原型详见附录 A.1。中断源信息详见附录 B。

警告:

已注册的回调函数只在 RAM 保持供电的睡眠模式期间保留。如果在睡眠过程中 RAM 停止供电并且需要中断，回调函数必须在唤醒调用 **u32AHI_Init( )**之前重新注册。

第4章 模拟外设

本章描述使用集成外设 API 来控制模拟外设。

JN516x 微控制器有 2 种类型的模拟外设：

- 模数转换器 [ADC]；
- 比较器；

模拟外设的中断在第 4.4 节中描述。

4.1 ADC

JN516x 微控制器包含一个 10 位的模数转换器（ADC）。ADC 采样一个模拟输入信号来产生输入电压的数字表示值。它瞬时采样输入电压，在将其转换成一个 10 位二进制值时把这个电压保存起来（保存到一个电容中）。整个采样/转换时间被称为转换时间。

ADC 可以周期采样来产生一个数字值序列，指示一段时间内输入电压的状态。采样事件发生的速度被称为采样频率。根据 Nyquist 采样定理，采样频率必须至少是测得的输入信号最高频率的 2 倍。如果输入信号包含大于采样频率一半的频率，这些频率就会混淆。为了防止混淆，ADC 输入应该连接一个低通滤波器来去除比采样频率一半更大的频率。

ADC 可以转换一个外部源、片内温度传感器和内部电压监控器（见下面的描述）的模拟输入。输入电压范围可以选择为 0~某个参考电压或者 0~该参考电压的 2 倍（见下面的描述）。

注:

当使用的 ADC 输入与一个 DIO 共用一个引脚时，相应的 DIO 应该配置成一个上拉禁能的输入（参考第 5.1.1 节和第 5.1.3 节）。

使用 ADC 时，调用的第一个模拟外设函数必须是 **vAHI_ApConfigure( )**，这个函数允许配置下列特性：

- **时钟：**

  ADC 的时钟输入由外设时钟提供，通常是 16MHz（关于系统时钟的选择见第 3.1 节），该频率被分频。目标频率使用 **vAHI_ApConfigure( )**来选择。推荐的 ADC 目标频率是 500kHz。

- **采样间隔和转换时间：**

  采样间隔确定了执行转换之前ADC捕获模拟输入电压的时间，实际上，捕获时间为这个采样间隔的 3 倍（见图 4.1）。采样间隔设置成ADC时钟周期的倍数（2x、4x、6x或8x），这个倍数使用函数**vAHI_ApConfigure( )**来选择，通常设置成 2x，详细信息请参考微控制器的数据手册。

  执行后面的转换所允许的时间为 13 个时钟周期。因此，采样和转换的总时间（转换时间）由下面的等式决定：

  $$[(3\times 采样间隔)+13]\times 时钟周期$$

  具体的图形演示请见图 4.1。

- **参考电压：**

  允许的模拟输入电压范围相对于参考电压 V_{ref} 来定义，参考电压来自于器件内部或外部。V_{ref} 源使用函数 **vAHI_ApConfigure( )**来选择。

  输入电压范围可以选择 0~V_{ref} 或 0~2V_{ref}，使用函数 **vAHI_AdcEnable( )**来选择，见后

面的描述。

- **稳压器：**

  为了最大限度地降低 ADC 的数字噪声，器件电压通过一个稳压器提供，稳压器的电源是模拟电源 VDD1。稳压器可以使用 **vAHI_ApConfigure( )**来使能/禁能。稳压器一旦被使能，就要等它稳定下来，可以使用函数 **bAHI_APRegulatorEnabled( )**来检查稳压器是否准备好。

- **中断：**

  中断可以使能，使得每次转换后都能产生中断（E_AHI_DEVICE_ANALOGUE 类型的中断）。这对于 ADC 的连续（周期性）转换特别有用。中断可以使用 **vAHI_ApConfigure( )**来使能/禁能。模拟外设的中断处理请见第 4.4 节的描述。

然后，ADC 必须使用函数 **vAHI_AdcEnable( )**进一步配置和使能（但不启动）。这个函数允许配置下列特性：

- **输入源：**

  ADC 的输入可以来自 6 个多路复用源中的一个（包含 4 个外部引脚（DIO））、一个片内温度传感器和一个内部电压监控器。输入使用 **vAHI_AdcEnable( )**来选择。

- **输入电压范围：**

  允许的模拟输入电压范围相对于参考电压 V_{ref} 来定义。输入电压范围可以选择 0~V_{ref} 或 0~2V_{ref}（这个范围以外的输入电压会导致饱和的数字输出），使用函数 **vAHI_AdcEnable( )**来选择。

- **转换模式：**

  ADC 可以配置成在下列模式下执行转换：

  - **单触发模式：**支持单次转换；
  - **连续模式：**重复执行转换；
  - **累加模式：**执行指定次数的转换，结果累加到一起。

  单触发模式或连续模式可以使用函数**vAHI_AdcEnable( )**来选择。在所有这 3 种模式的情况下，单次转换的转换时间都是*[(3×采样间隔)+13] ×时钟周期*，见图 4.1。在连续模式和累加模式的情况下，转换时间过后启动下次转换，采样频率是转换时间的倒数。

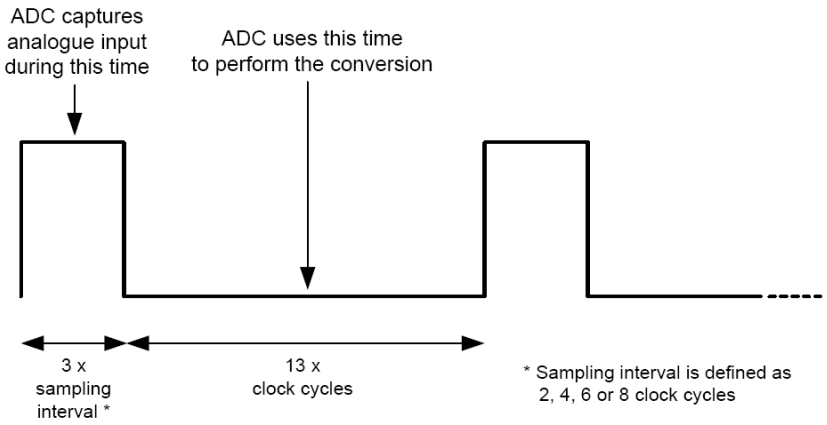

图 4.1 ADC 采样

一旦通过先调用函数 **vAHI_ApConfigure( )**，再调用函数 **vAHI_AdcEnable( )**配置完 ADC

之后，转换就在下面其中一种模式下启动。ADC 在这些模式下的操作在后面各小节中描述。

- 单触发模式
- 连续模式
- 累加模式

请注意，只有 ADC 可以产生模拟外设中断（E_AHI_DEVICE_ANALOGUE 类型）。这些中断由一个通过 **vAHI_APRegisterCallback( )**注册的用户定义的回调函数来处理。关于模拟外设中断处理的更多信息请参考第 4.4 节。

4.1.1 单触发模式

在单触发模式下，ADC 执行一次转换然后终止。要工作在这种模式下，必须在 ADC 使用 **vAHI_AdcEnable( )**使能时就已经选择了该模式，然后才能使用函数 **vAHI_AdcStartSample( )** 来启动转换。

可以使用下面其中一种方法来检测转换的结束：

- 在转换结束时产生中断。这种情况下，模拟外设中断必须已经在函数 **vAHI_ApConfigure( )**中使能了；
- 可以使用函数 **bAHI_AdcPoll( )**来检查转换是否已经结束。

只要执行完转换，转换结果就可以使用函数 **u16AHI_AdcRead( )**来获得。

4.1.2 连续模式

连续模式下，ADC 不停地执行重复转换（直至终止）。要在这种模式下工作，连续模式必须在 ADC 使用 **vAHI_AdcEnable( )** 使能时就已经选择好了，然后才能使用函数 **vAHI_AdcStartSample( )**来启动转换。

连续模式下的采样频率是转换时间的倒数，转换时间由下面的等式得出：

$$转换时间 = [(3 \times 采样间隔)+13] \times 时钟周期$$

可以使用下面其中一种方法来检测转换的结束：

- 在转换结束时产生中断。这种情况下，模拟外设中断必须已经在函数 **vAHI_ApConfigure( )**中使能了；
- 可以使用函数 **bAHI_AdcPoll( )**来检查转换是否已经结束。

只要执行完一次转换，转换结果就可以使用函数 **u16AHI_AdcRead( )**来获得。下次转换完成之前都可以通过这个函数来读取结果。

转换可以使用函数 **vAHI_AdcDisable( )**来终止。

4.1.3 累加模式

在累加模式下，ADC 执行固定次数的转换然后终止。这些转换的结果累加到一起来取平均值。要在这种模式下工作，转换必须使用函数 **vAHI_AdcStartAccumulateSamples( )**来启动。转换的次数在这个函数中选择，可以选择 2、4、8 或 16 次。

注：

当 ADC 在累加模式中启动时，**vAHI_AdcEnable( )**中选择的转换模式被忽略。

累加模式下的采样频率是转换时间的倒数，转换时间由下面的等式得出：

$$转换时间 = [(3 \times 采样间隔)+13] \times 时钟周期$$

可以使用下面其中一种方法来检测全部转换的结束：

- 在转换结束时产生中断。这种情况下，模拟外设中断必须已经在函数 **vAHI_ApConfigure( )**中使能了；
- 可以使用函数 **bAHI_AdcPoll( )**来检查转换是否已经结束。

只要执行完转换，累加的结果就可以使用函数 **u16AHI_AdcRead( )**来获得。请注意，这个函数提供的是各次转换结果的和。取平均值的计算必须由应用程序来完成（除以转换的次数）。

转换可以随时使用函数 **vAHI_AdcDisable( )**来终止。

4.2 带DMA引擎的ADC（采样缓冲模式）

本节描述 ADC 和 JN516x 器件上的 DMA（直接存储器存取）引擎一起使用的工作模式。在这个模式下：

- 在固定的时间间隔产生 ADC 10 位数据采样，16 位的采样结果传输到 RAM 的缓冲区中，数据传输和存储都由 DMA 引擎执行，不需要 CPU 的干预；
- DMA 引擎正在处理数据传输和存储时 CPU 可以执行其他任务。CPU 只需要启动 ADC 转换并在中断出现时处理缓冲区中的结果；
- ADC 采样可以在不同的模拟外设之间多路复用。

这种使用 ADC 的方法称为‘采样缓冲模式’。

ADC 采样以可配置的速度产生，而且可以使用一个片内定时器（定时器 0、1、2、3 或 4）来定时。

当缓冲区已经装满足够多的数据导致产生中断时，CPU 上运行的应用程序就可以服务缓冲区。应用程序必须注册一个回调函数来服务这个中断。

4.2.1 采样缓冲模式的前期准备

在使能和启动采样缓冲模式之前必须做好下列准备：

- 必须调用函数 **vAHI_ApConfigure( )**来完成 ADC 的初始配置（包括转换的时钟频率、转换的采样间隔、输入的参考电压，稳压器的使用和中断的使用），见第 4.2 节中其他 ADC 模式的描述；
- 必须设置和启动一个 JN516x 定时器触发重复转换。请注意以下几点：
  - 这个定时器可以是定时器 0~4 中的任何一个（采样缓冲模式使能和启动时，需要的定时器稍后指定，见第 4.2.2 节）；
  - 这个定时器不需要 DIO，DIO 可以用作其它用途（见第 7.2.1 节）；
  - 不需要定时器中断，而且当调用 **vAHI_TimerEnable( )**时应当为这个定时器禁能中断（见第 7.2.2 节）；
  - 必须使用 **vAHI_TimerStartRepeat( )**在‘定时器重复’模式下配置和启动定时器（见第 7.3.1 节）。
- 一个用来处理采样缓冲模式下所产生中断的用户定义的回调函数必须使用 **vAHI_APRegisterCallback( )**来注册，见第 4.4 节的描述（当采样缓冲模式使能和启动时，需要的中断模式稍后指定，见第 4.2.2 节）。

4.2.2 采样缓冲模式操作

一旦采样缓冲模式已经像第 4.2.1 节中描述的那样准备好（ADC 已配置，定时器已启动，回调函数已注册），这个模式下的操作可以使用 **bAHI_AdcEnableSampleBuffer( )**进一步配置和启动。下面的配置必须在这个函数调用中执行：

- 需要的 JN516x 定时器必须指定为定时器 0~4 中的一个；
- 输入电压范围必须指定成 0~V_{ref} 或 0~2V_{ref}，参考电压 V_{ref} 已经在 **vAHI_ApConfigure( )** 调用中设定好了；
- 必须提供一个位图表，在位图表中指定了这个 ADC 模式下将要复用的模拟输入源。可能的模拟输入源包括 4 个外部引脚（DIO）、一个片内温度传感器和一个内部电压监控器；
- 接收数据的 RAM 缓冲区必须完全设定好：
  - 指向缓冲区起始处的指针；
  - 缓冲区的大小（在 16 位采样中，缓冲区高达 2047）；
  - 缓冲区已满时是否回到起始处；
- 使用的 DMA 中断模式必须设置成下面其中一种情况：
  - 缓冲区填充到中间点时产生中断；
  - 缓冲区满时产生中断；
  - 缓冲区回到起始处时产生中断；

如果这个模式下的操作是连续的（缓冲区环回），该模式就可以使用函数 **vAHI_AdcDisableSampleBuffer( )**来停止。

采样缓冲模式操作各方面的注意事项（输入复用、缓冲区环回和 DMA 中断）在下面描述。

输入复用

采样缓冲模式允许多达 6 个模拟输入多路复用。这些输入包括 4 个外部输入（ADC1-4，对应 DIO 引脚），一个片内温度传感器和一个内部电压监控器。需要的复用输入在 **bAHI_AdcEnableSampleBuffer( )**调用中通过一个位图表来指定。

所选输入的 16 位采样在每个定时器触发下产生。这些采样以下面的顺序产生和保存到缓冲区中：

1）外部输入 ADC1

2）外部输入 ADC2

3）外部输入 ADC3

4）外部输入 ADC4

5）温度传感器

6）电压监控器

缓冲区回环

在 **bAHI_AdcEnableSampleBuffer( )**调用中，RAM 缓冲区可以配置成回环，也就是说，当缓冲区已满时数据继续重新从缓冲区起始处写入（如果配置了就会产生中断）。

如果没有选择缓冲区回环，只要缓冲区已满转换就自动停止（如果配置了就会产生中断）。

DMA 中断

DMA 中断用来通知应用 RAM 缓冲区的状态。在下列情况下可以产生 DMA 中断：

- 缓冲区已经半满；
- 缓冲区已经全满；
- 缓冲区已经全满，并且下个数据将写入缓冲区起始处（假设缓冲区回环已经使能）。

上面的一个或多个中断条件可以在 **bAHI_AdcEnableSampleBuffer( )**调用中选择。这些中

断必须由使用 **vAHI_APRegisterCallback( )**注册的用户定义的回调函数来服务。

4.3 比较器

JN516x 微控制器包含一个比较器（编号为 1）。

比较器可以用来比较 2 个模拟输入。当比较器输入之间的算术差改变方向（正变为负或负变为正）时比较器改变两个状态的数字输出（高到低或低到高）。当与一个固定参考输入相比较时，比较器可以用作一个测量随时间变化的模拟输入的频率的基准。

一个模拟输入承载着要监测的外部信号。输入电压必须总是保持在 0V~V_{dd}（芯片电源电压）的范围以内。这个外部信号可以来自比较器的‘正相’引脚（COMP1P）或‘反相’引脚（COMP1M），与一个参考信号进行比较，参考信号可以来自外部或内部：

- 参考信号来自外部的其他比较器引脚（COMP1P 或 COMP1M），该引脚当前未用于监控的输入信号；
- 参考信号来自内部的参考电压 V_{ref}（V_{ref} 的来源使用函数 **vAHI_ApConfigure( )**来选择）。

输入和参考信号通过函数 **vAHI_ComparatorEnable( )**从上述选项中进行选择，这个函数用来配置和使能比较器。

注 1:

默认情况下，比较器在低功耗模式下使能。更详细的信息请参考第 4.3.2 节。

注 2:

ADC 正在运行时调用 **vAHI_ComparatorEnable( )**可能导致 ADC 结果出错。因此，如果需要的话，这个函数应该在启动 ADC 之前调用。

注 3:

当一个比较器引脚被使用时，相应的 DIO 应该配置成上拉禁能的输入（参考第 5.1.1 节和第 5.1.3 节）。

比较器有两个可能的状态：高或低。比较器状态由 2 个模拟输入电压的相对值来决定，也就是说，由电压的瞬时电压 V_{sig} 和参考电压 V_{refsig} 来决定。两者的关系如下所述：

$$V_{sig} > V_{refsig} \rightarrow 高$$

$$V_{sig} < V_{refsig} \rightarrow 低$$

或根据两者之间的差来决定：

$$V_{sig} - V_{refsig} > 0 \rightarrow 高$$

$$V_{sig} - V_{refsig} < 0 \rightarrow 低$$

因此，随着信号电平根据时间的变化，当 V_{sig} 上升到高于 V_{refsig} 或下降到低于 V_{refsig} 时，比较器结果的状态发生改变。这样，V_{refsig} 用作阈值来评估 V_{sig} 的变化。

实际上，这种方法对于造成比较器状态虚假变化的模拟输入信号中的噪声很敏感。这时可以使用 2 个不同的阈值的改善：

- 一个上限阈值 V_{upper}，用于低到高的跳变；
- 一个下限阈值 V_{lower}，用于高到低的跳变。

阈值 V_{upper} 和 V_{lower} 这样定义，使得它们高出和低于参考信号 V_{refsig} 的量是相同的，这个量就称为滞后电压 V_{hyst}。

即，

$$V_{upper} = V_{refsig} + V_{hyst}$$

$$V_{lower} = V_{refsig} - V_{hyst}$$

滞后电压使用函数 **vAHI_ComparatorEnable( )**来选择。它可以设置成 0、5、10 或 20mV。所选的滞后电平应该大于输入信号中的噪声电平。

对于参考信号电压V_{refsig}是一个常量时比较器的 2 个阈值机制如下面的图 4.2所示。

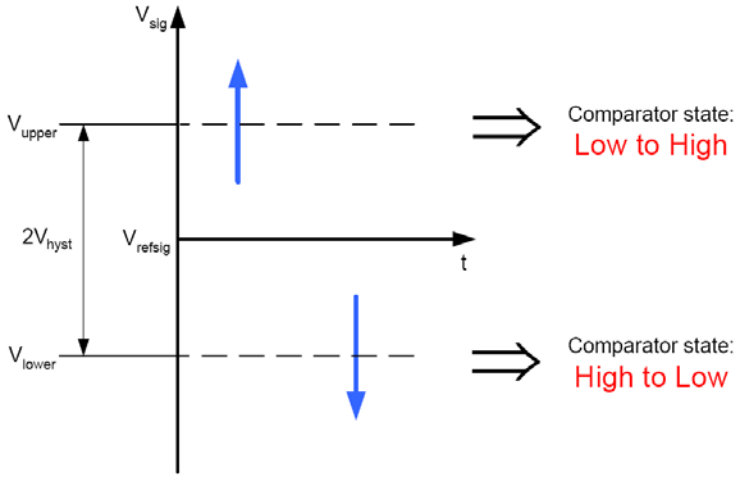

图 4.2 比较器的上限和下限阈值

请注意，在比较器输入的改变和比较器报告的最终状态之间有一个时间延迟。

vAHI_ComparatorEnable( )不仅能配置比较器，还能启动比较器的操作。比较器的当前状态（高或低）可以使用函数 **u8AHI_ComparatorStatus( )**随时获得。比较器可以使用函数 **vAHI_ComparatorDisable( )**随时停止。

4.3.1 比较器中断和唤醒

比较器允许在输出出现低到高或高到低的跳变时产生中断。中断只能在出现一个方向（不是两个方向）的跳变时产生。中断可以使用函数 **vAHI_ComparatorIntEnable( )**来使能。这个函数用来使能/禁能比较器中断和选择触发中断的跳变方向。

重点:

比较器中断是系统控制器中断，不是模拟外设中断。因此，必须由通过 **vAHI_SysCtrlRegister Callback( )**注册的一个回调函数来处理。

比较器中断可以用作一个将节点从睡眠中唤醒的信号，又称为‘唤醒中断’。为了使用这个特性，像上面描述的一样，中断必须使用 **vAHI_ComparatorIntEnable( )**来配置和使能。请注意，在睡眠期间，参考信号 V_{refsig} 来自一个外部源，由‘反相’引脚 COMP1M 或‘正相’引脚 COMP1P（在唤醒期间使用其中一个）提供。唤醒中断状态可以使用函数 **u8AHI_ComparatorWakeStatus( )**来检查。

4.3.2 比较器低功耗模式

比较器能工作在低功耗模式，在该模式下它只消耗 0.8μA 的电流，而工作在标准功耗模式下需要消耗 73μA 的电流。比较器低功耗模式可以使用函数 **vAHI_ComparatorLowPower Mode( )**来使能/禁能。

比较器使用 **vAHI_ComparatorEnable( )**来配置和启动时工作在标准功耗模式。为了使比较器工作在低功耗模式，还必须调用函数 **vAHI_ComparatorLowPowerMode( )**。

低功耗模式有助于最大限度地降低使用能量采集的设备所消耗的电流。为了使能量消耗保

持最佳，低功耗模式中在睡眠过程中也自动使能。低功耗模式的缺点是比较器响应变慢，即，比较器输入改变和比较器报告的最终状态之间的延时时间更长。但是，如果响应时间不是很重要的情况下通常应该使用低功耗模式。

4.4 模拟外设中断

模拟外设中断（类型 E_AHI_DEVICE_ANALOGUE）只由 ADC 产生，比较器产生系统控制器中断。模拟外设中断在函数 **vAHI_ApConfigure( )** 中使能，由一个使用函数 **vAHI_APRegisterCallback( )**注册的用户定义的回调函数来处理。关于回调函数原型的详情请参考附录 A.1。调用回调函数时中断自动清除。

警告:

已注册的回调函数只在 RAM 保持供电的睡眠模式期间保留。如果 RAM 在睡眠期间掉电而又需要中断，回调函数必须在唤醒调用 **u32AHI_Init( )**之前重新注册。

确切的中断源由 ADC 工作模式（单触发、连续和累加模式）决定：

- 在单触发和连续模式下，每次转换后都产生一个‘捕获’中断；
- 在累加模式下，最终累加的结果可用时产生一个‘累加’中断。

一旦 ADC 结果变得可用，ADC 结果就可以通过在回调函数中调用函数 **u16AHI_AdcRead( )** 来获得。

第5章 数字输入/输出（DIO）

本章描述使用集成外设 API 的函数来控制数字输入/输出（DIO）。

JN516x 微控制器有 20 个 DIO 线，编号为 0~19。每个引脚可以单独配置成输入或输出。但是，DIO 引脚和下列片内外设/性能共用同一个引脚：

- ADC
- 比较器
- UART
- 定时器
- 2 线串行接口（SI）
- 串行外设接口（SPI）
- 天线分集
- 脉冲计数器

相应的外设/性能使能时共用的 DIO 不能使用。关于共用引脚的详情请参考微控制器的数据手册。

复位时，所有外设禁能，DIO 配置成输入。

除了正常操作以外，DIO 配置成输入时可以用来产生中断和将器件从睡眠中唤醒，见第 5.2 节。请注意，DIO 触发的中断是系统控制器中断，由通过 **vAHI_SysCtrlRegisterCallback( )**注册的一个回调函数来处理，见第 3.5 节。

注:

除了 DIO 之外，JN516x 器件有 2 个数字输出（DO0 和 DO1）。这些输出的配置在第 5.3 节中描述。

5.1 使用DIO

本节描述了如何使用集成外设 API 函数来配置和访问 DIO。

5.1.1 设置DIO的方向

DIO 可以使用函数 **vAHI_DioSetDirection( )**单独配置成输入和输出，默认情况下它们全部是输入。如果调用 **vAHI_DioSetDirection( )**时 DIO 和一个片内外设共用一个引脚并且外设正在使用该引脚，指定的 DIO 输入/输出设置不会立刻生效，而是等到外设禁能之后才生效。

5.1.2 设置DIO输出

配置成输出的 DIO 可以使用函数 **vAHI_DioSetOutput( )**单独设置成接通（高）和断开（低）。输出状态在一个 32 位的图表中设置，在图表中每个 DIO 由一个位来代表（位 0 – 19 对应 DIO0-19）。请注意以下几点：

- 配置成输入的 DIO 不受这个函数的影响，除非这些 DIO 后来通过调用 **vAHI_DioSetDirection( )**设置成输出。然后它们才会采纳 **vAHI_DioSetOutput( )**中设置的输出状态；
- 如果在调用 **vAHI_DioSetOutput( )**时一个共用的 DIO 正被一个片内外设使用，指定的 DIO 接通/断开设置就不会立刻生效，而是等到这个外设禁能之后才生效。

一组 8 个连续 DIO 可以用来并行输出一个字节，DIO0-7 或 DIO8-15 就可以这么做，在这里位 0 或 8 作为字节的最低有效位。DIO 组和输出字节可以使用函数 **vAHI_DioSetByte( )**来指定。所选组中的所有 DIO 必须事先已经配置成输出，见第 5.1.1 节。

5.1.3 设置DIO上拉

每个 DIO 都有一个关联的上拉电阻。‘上拉’的目的是在 DIO 没有连接外部负载时防止引脚状态‘悬浮’，也就是说，上拉使能时，在没有外部负载的情况下（或者存在一个弱外部负载的情况下）上拉将引脚拉高（接通）。所有 DIO 的上拉可以使用函数 **vAHI_DioSetPullup( )**来使能/禁能，默认情况下所有上拉使能。此外，如果调用 **vAHI_DioSetPullup( )**时一个共用 DIO 正在被一个片内外设使用，指定的 DIO 上拉设置就被运用，但 DIO 连接一个外部 32kHz 晶体的情况除外（见第 3.1.4 节）。

注:

DIO 上拉设置在睡眠期间保持。如果不需要上拉，将 DIO 上拉禁能（在睡眠或正常操作期间）可以节省功耗。

5.1.4 读取DIO

DIO 的状态可以使用函数 **u32AHI_DioReadInput( )**来获取。这个函数将返回所有 DIO 的状态，与 DIO 配置成输入或输出以及是否被外设使用无关。

一组 8 个连续 DIO 可以用来并行输入一个字节，DIO0-7 或 DIO8-15 就可以这么做，在这里位 0 或 8 作为字节的最低有效位。DIO 组的输入字节可以使用函数 **u8AHI_DioReadByte( )** 来获取。组中所有 DIO 都必须事先已经配置成输入，见第 5.1.1 节。

5.2 DIO中断和唤醒

配置成输入的 DIO 可以用来产生系统控制器中断。这些中断可以在微控制器正在睡眠时用来唤醒微控制器。集成外设 API 包括一组‘DIO 中断’函数和一组‘DIO 唤醒’函数，但这些函数的作用是一样的（因为它们访问硬件中相同的寄存器位）。这两组函数的使用在下面各小节中描述。

警告:

由于‘DIO 中断’和‘DIO 唤醒’函数访问相同的 JN516x 寄存器位，用户必须确保两组函数在应用代码中不冲突。

5.2.1 DIO中断

输入 DIO 状态的改变可以用来触发中断。

首先，应该使用函数 **vAHI_DioInterruptEdge( )**为单个 DIO 选择触发中断的输入信号跳变（低到高或高到低），默认情况下是低到高的跳变。然后使用函数 **vAHI_DioInterruptEnable( )** 使能相关 DIO 的中断。

DIO 引脚的中断状态可以随后使用函数 **u32AHI_DioInterruptStatus( )**来获取，即，这个函数可以用来确定一个 DIO 是否产生了中断。当 DIO 中断禁能进而不产生时这个函数用来轮询 DIO 的中断状态。

注:

如果 DIO 中断使能，用户应当在通过 **vAHI_SysCtrlRegisterCallback( )**注册的回调函数中加入 DIO 中断的处理。

5.2.2 DIO唤醒

DIO 可以用来将微控制器从睡眠（包括深度睡眠）或休眠模式中唤醒。配置成输入的任何 DIO 引脚可以用于唤醒，DIO 的状态变化会触发唤醒中断。

首先，应该使用函数 **vAHI_DioWakeEdge( )**为单个 DIO 选择触发唤醒中断的输入信号跳变

（低到高或高到低），默认情况下是低到高的跳变。然后使用函数 **vAHI_DioWakeEnable( )**使能相关 DIO 引脚的唤醒中断。

DIO 引脚的唤醒状态可以随后使用函数 **u32AHI_DioWakeStatus( )**来获取，即，这个函数可以用来确定一个 DIO 是否导致产生了唤醒事件。请注意，唤醒时用户必须在 **u32AHI_Init( )**之前调用这个函数，因为函数 **u32AHI_Init( )**会清除所有挂起的中断。

注 1:

代替调用函数 **u32AHI_DioWakeStatus( )**，用户也可以在通过 **vAHI_SysCtrlRegisterCallback( )**注册的回调函数中确定唤醒中断源。

注 2:

从深度睡眠中唤醒时，函数 **u32AHI_DioWakeStatus( )**不指示 DIO 唤醒源，因为器件已经结束了全部的复位。当从睡眠中唤醒时，如果在器件启动时出现多个 DIO 事件，这个函数可以指示多个唤醒源。

注 3:

如果使用 JenNet 协议，从睡眠中唤醒时不调用 **u32AHI_DioWakeStatus( )**来获取 DIO 中断状态。唤醒时 JenNet 会在内部调用 **u32AHI_Init( )**并清除中断状态，然后再将控制权交给应用程序。如果需要，必须使用系统控制器回调函数来获取中断状态。

5.3 配置数字输出（DO）

JN516x 器件有 2 个引脚 DO0 和 DO1，器件唤醒时它们用作通用数字输出。DO 引脚与 SPI 主机和定时器 2、3 共用引脚。

这些引脚可以配置如下：

- **bAHI_DoEnableOutputs( )**可以用来使能（或禁能）DO 引脚用作通用数字输出，默认情况下，DO 引脚上电时被禁止用作通用数字输出；
- **vAHI_DoSetDataOut( )**可以用来将这两个 DO 引脚的输出状态设置成接通或断开（可以是任何一种组合），默认情况下，输出状态上电时是接通的；
- **vAHI_DoSetPullup( )**可以用来将这两个 DO 引脚的上拉状态设置成接通或断开（可以是任何一种组合），默认情况下，上拉上电时是使能的。

在睡眠过程中 DO 引脚的状态不保持，它不能用来将器件从睡眠中唤醒。复位、睡眠期间以及从睡眠模式唤醒时，DO 引脚恢复为禁止用作上拉使能的通用输出。

第6章 UART

本章描述使用集成外设 API 来控制 UART（通用异步收发器）。

JN516x 微控制器有 2 个 UART，称为 UART0 和 UART1，它们可以单独使能。这些 UART 兼容 16550，在高达 4Mbps 的可编程波特率下可以用于串行数据的输入/输出。

注:

这里描述的 UART 操作假设外设时钟运行在 16MHz，时钟来自一个外部晶体振荡器，见第 3.1 节。不建议用户在任何其他时钟下运行 UART。

6.1 UART信号和引脚

UART 使用以下信号与一个外部设备相连：

- 发送数据（TxD）输出 —— 连接到外部设备的 RxD
- 接收数据（RxD）输入 —— 连接到外部设备的 TxD
- 请求发送（RTS）输出 —— 连接到外部设备的 CTS
- 清除发送（CTS）输入 —— 连接到外部设备的 RTS

如果 UART 只使用信号 RxD 和 TxD，就表明 UART 工作在 2 线模式（见第 6.2.1 节）。如果 UART 使用上面全部 4 个信号，就表明 UART 工作在 4 线模式并执行流控制（见第 6.2.2 节）。在 JN516x 器件上：

- UART0 可以工作在 4 线模式（默认情况）或 2 线模式；
- UART1 可以工作在 2 线模式（默认情况）或 1 线（仅发送）模式；

上面信号使用的默认引脚与DIO共用，如下面的表 6.1所示：

表 6.1 UART 信号使用的默认 DIO

信号	UART0 使用的 DIO*	UART1 使用的 DIO
CTS	DIO4	-
RTS	DIO5	-
TxD	DIO6	DIO14
RxD	DIO7	DIO15

* UART0 使用的引脚还可以用来连接一个 JTAG 仿真器进行调试。

UART0 信号可以使用函数 **vAHI_UartSetLocation( )**从 DIO4-7 移至 DIO12-15。UART1 信号可以使用相同的函数从 DIO14 和 DIO15 分别移至 DIO11 和 DIO9。如果需要这个函数，必须在 UART 使能之前调用它。

6.2 UART操作

UART 的发送和接收通路每个都有一个 FIFO 缓冲区，它允许和外部设备执行多字节的串行传输：

- TxD 引脚连接到发送 FIFO；
- RxD 引脚连接到接收 FIFO；

FIFO 包含在 RAM 中，由应用来定义。每个 FIFO 的大小可以从 16 个字节直至 2047 个字节。

在本地设备上，CPU 可以一次性向 FIFO 写入/读出一个字节和一个数据块。读/写通路互相

独立，因此发送和接收单独出现。由一个 DMA（直接存储器存取）引擎来处理 FIFO 和 TxD/RxD 之间的数据移动，不需要 CPU 的参与。

下面的图 6.1描绘了基本的UART设置。

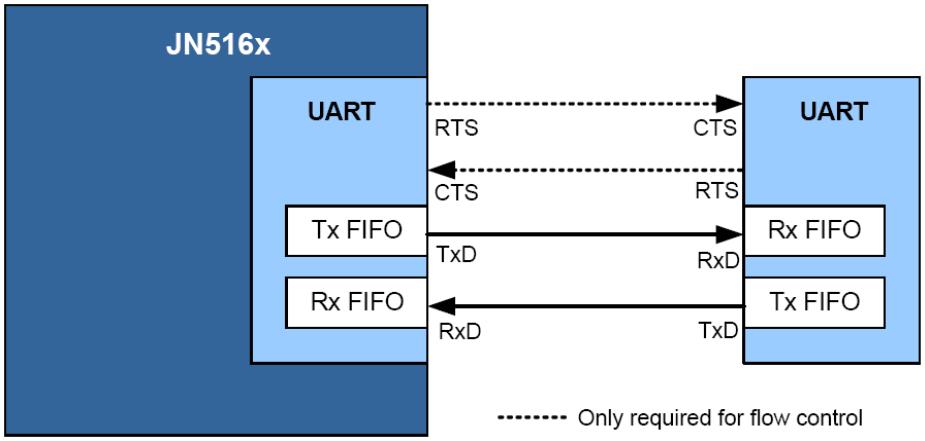

图 6.1 UART 连接

两个 UART 都可以工作在 2 线模式，UART0 还可以工作在 4 线模式，UART1 还可以工作在 1 线模式。这些模式在下面各小节中描述。

6.2.1 2 线模式

在 2 线模式下，UART 只使用信号线 TxD 和 RxD。只要发送设备方便（例如，当发送 FIFO 包含一些数据时），可以不做任何通知就发送数据。数据还也可以根据发送设备的方便不做任何通知就接收。这可能会产生问题和导致数据丢失，例如，接收设备可能在接收 FIFO 中没有足够的空间来接收发送过来的数据。

6.2.2 4 线模式（带流控制）（仅适用UART0）

在 4 线模式下，UART0 使用信号线 TxD、RxD、RTS 和 CTS，允许执行流控制，保证发送的数据总是能接收到。流控制的一般原理在下面描述。

RTS 和 CTS 线是标志，用来指示数据在设备之间安全传输。一个设备上的 RTS 线连接到另一个设备的 CTS 线。

目标设备命令源设备应该何时给它发送数据，如下所述：

- 当目标设备准备好接收数据时，它激活 RTS 线来请求源设备发送数据。这种情况可能是目标设备的接收 FIFO 填充水平（fill-level）降至低于一个预定义好的水平，并且 FIFO 能够接收更多的数据；
- 源设备将目标设备 RTS 线的激活看成是自己 CTS 线的激活，然后将数据从发送 FIFO 中发送出去。

流控制操作如下面图 6.2的所示。

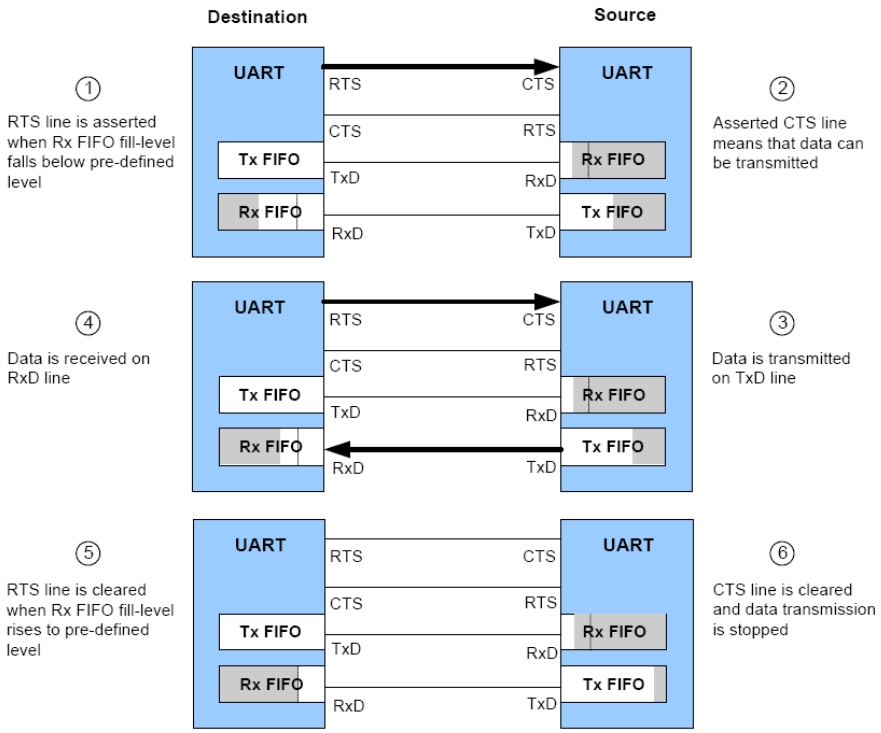

图 6.2 UART 流控制的示例（仅适用于 UART0）

集成外设 API 提供了函数来控制和监控 RTS/CTS 线，允许应用手动执行流控制算法。在实际操作中，手动流控制对于繁忙的 CPU 来说是一个负担，特别是 UART 正在一个高波特率下运行的时候。为此，API 提供一个自动流控制选择，在自动流控制中，RTS 线的状态直接由目标设备的接收 FIFO 填充水平来控制。使用集成外设 API 来实现手动和自动流控制请参考第 6.5 节的描述。

6.2.3 1 线模式（仅适用UART1）

在 1 线模式下，UART1 根据发送设备方便，在不通知的情况下使用 TxD 线发送数据。该模式下不接收数据，也不使用 RxD 线（因此，相关的 DIO 引脚可以用作其他用途）。

6.3 配置UART

本节描述使用 UART 传输串行数据前对 UART 的各种配置。

6.3.1 使能UART

UART 使用函数 **bAHI_UartEnable( )**来使能，默认情况下，该函数使能 UART0 进入 4 线模式，或使能 UART1 进入 2 线模式。这个函数必须是调用的第一个 UART 函数，除非用户希望在非默认模式下使用 UART：

- 如果用户希望在 2 线模式（无流控制）下使用 UART0，为了释放用于流控制 RTS 和 CTS 线的 DIO 的控制权，用户要首先调用 **vAHI_UartSetRTSCTS( )**;
- 如果用户希望在 1 线模式（只发送）下使用 UART1，为了释放用作 RxD 的引脚的控制权，用户要首先调用 **vAHI_UartTxOnly( )**。

函数 **bAHI_UartEnable( )**还允许配置 UART 发送和接收通路的 FIFO 缓冲区。应用将每个缓冲区定义成 RAM 中的一部分，必须设定每个缓冲区的起始地址和大小（用字节表示）。缓冲区最大为 2047 个字节，最小为 16 个字节。

注:

函数 **vAHI_UartEnable( )**（返回 void，而不是布尔值）也可以向后兼容 JN514x 微控制器的应用代码。

6.3.2 设置波特率

下列函数用来设置 UART 的波特率：

- **vAHI_UartSetBaudRate( )**

  这个函数允许设置成下面其中一个标准波特率：4800、9600、19200、38400、76800 或 115200bps。

- **vAHI_UartSetBaudDivisor()**

  这个函数允许指定一个 16 位整数分频值（*Divisor*），用来从一个 1MHz 的频率中获得波特率，由下式给出：

$$\frac{1*10^6}{Divisor}$$

- **vAHI_UartSetClocksPerBit()**

  与单独使用 **vAHI_UartSetBaudDivisor( )**相比，使用这个函数可以获得一个更精确的波特率。**vAHI_UartSetBaudDivisor( )**得到的分频值与 **vAHI_UartSetClocksPerBit( )** 的 8 位整型参数（Cpb）联合起来从 16MHz 的外设时钟获得一个波特率，由下式给出：

$$\frac{16*10^6}{Divisor*(Cpb+1)}$$

  根据上面的公式，可以获得推荐的最高波特率 4Mbps（*Divisor* =1，*Cpb* = 3）。

注:

必须要么调用 **vAHI_UartSetBaudRate( )**，要么调用 **vAHI_UartSetBaudDivisor( )**，两者不可同时调用。在使用过程中，必须先调用 **vAHI_UartSetBaudDivisor( )**，再调用 **vAHI_UartSetClocks PerBit( )**。

6.3.3 设置其他UART特性

除了设置 UART 的波特率（见第 6.3.2 节的描述），还需要配置一些其他 UART 特性。这些特性采用函数 **vAHI_UartSetControl( )**来设置，包括以下内容：

- 可以选择奇偶校验应用到传输数据中，并且可以选择奇偶校验的类型（奇校验或偶校验）；
- 一个数据字的长度可以设置为 5、6、7 或 8 个位。这是每个发送‘字符’的位数，通常设置为 8（一个字节）；
- 停止位的数量可以设置为 1、1.5 或 2；
- 可以配置 RTS 线的初始状态（设置或清除），这只能在 UART0 工作在 4 线模式下使用（见第 6.3.1 节）。

6.3.4 使能中断

UART 中断可以在多种条件下产生。中断可以使用函数 **vAHI_UartSetInterrupt( )**来使能和配置。可能的中断条件如下所述：

- **发送 FIFO 空**：发送 FIFO 已空（因此需要更多数据）；

- **接收数据可用**：接收 FIFO 装入的数据达到预先定义的级别，这个级别可以设置成 1、4、8 或 14 个字节。当 FIFO 填充水平再次降至低于预先定义的级别时清除中断；
- **超时**：当‘接收数据可用’中断使能时这个中断使能；如果所有下面的条件都存在时这个中断产生：
  - FIFO 中至少有一个字符；
  - 在一段时间间隔内没有字符进入 FIFO，而在这段时间内应该能够接收到至少 4 个字符；
  - 在一段时间间隔内没有从 FIFO 中读出数据，而在这段时间内应该能够读出至少 4 个字符。

  通过从接收 FIFO 中读取一个字符来清除超时中断和复位定时器。
- **接收线状态**：RxD 线上出现一个错误条件，例如，间断指示、帧错误、奇偶错误或超限运行；
- **Modem 状态**：UART0 工作在 4 线模式时检测到 CTS 线发生变化（例如，它已经被激活，指示远程设备已经准备好接收数据）。

UART 中断由一个回调函数来处理，这个函数必须使用函数 **vAHI_Uart0RegisterCallback( )** 或 **vAHI_Uart1RegisterCallback( )**来注册，具体由 UART（UART0 或 UART1）决定。关于 UART 中断处理的更多信息请参考第 6.8 节。

6.4 在 2 线模式下传输串行数据

在 2 线模式下，UART 只使用 RxD 和 TxD，不执行流控制。数据发送和接收分别在下面描述。

注 1:

对于 UART1，2 线模式是默认模式。

注 2:

为了在 2 线模式下运行 UART0，必须首先调用函数 **vAHI_UartSetRTSCTS( )**来释放用于流控制的 DIO 的控制权。这个函数必须在 **vAHI_UartEnable( )**之前调用。

6.4.1 发送数据（2 线模式）

调用下面其中一个函数来通过 UART 发送数据：

- **vAHI_UartWriteData( )**：这个函数可以用来写一个字节的数据到发送 FIFO。一旦使用，这个函数可以调用多次调用来排队数据字节进行发送；
- **u16AHI_UartBlockWriteData( )**：这个函数用来写一个数据字节块到发送 FIFO。这个函数将返回已经成功写入 FIFO 中的字节数。

在 FIFO 中，只要一个数据字节到达 FIFO 头部（假设 TxD 线空闲）就开始发送这个数据。FIFO 到 TxD 线的数据传输由 DMA 引擎自动处理。

下列方法可以用来提示应用调用 **vAHI_UartWriteData( )**或 **u16AHI_UartBlockWriteData( )**函数：

- 可以调用函数 **u16AHI_UartReadTxFifoLevel( )**来检查当前正在发送 FIFO 中等待的字符数（如果有足够的空闲空间，可以有更多的数据写入 FIFO 中）；
- 函数 **u16AHI_UartReadLineStatus( )**可以用来检查发送 FIFO 是否为空；
- 当发送 FIFO 变为空（即，开始发送 FIFO 的最后一个数据字节时）时产生一个中断。这个中断使用函数 **vAHI_UartSetInterrupt( )**来使能；

- 可以使用一个定时器来调度定期发送（前提是有数据可以发送）。

6.4.2 接收数据（2 线模式）

源设备发送的数据在 RxD 上接收。RxD 线到接收 FIFO 的本地数据传输由 DMA 引擎自动处理。目标应用可以使用下面其中一个函数来读取 FIFO 中的数据：

- **u8AHI_UartReadData( )**：这个函数可以用来从接收 FIFO 中读出一个字节的数据；
- **u16AHI_UartBlockReadData( )**：这个函数可以用来从接收 FIFO 中读出一个数据字节块。

可以使用下面的方法来提示应用调用 **u8AHI_UartReadData( )**或 **u16AHI_UartBlockReadData( )**函数：

- 可以调用函数 **u16AHI_UartReadRxFifoLevel( )**来检查当前接收 FIFO 中的字符数；
- 可以使用函数 **u8AHI_UartReadLineStatus( )**来检查接收 FIFO 是否包含可以读取的数据（或者接收 FIFO 是否为空）；
- 当接收 FIFO 包含某个数量的数据字节时产生中断；这个中断使用函数 **vAHI_UartSetInterrupt( )**来使能，在这个函数中必须将中断的触发级别设定为 1、4、8 或 14 个字节；
- 可以使用一个定时器来调度周期性读取接收 FIFO。在每次读取之前，可以使用 **u8AHI_UartReadLineStatus( )**或 **u16AHI_UartReadRxFifoLevel( )**来检查 FIFO 中数据的存在。

注:

当‘接收数据可用’中断使能（见上面的描述）时，接收 FIFO 的‘超时’中断也使能。关于这个中断更详细的信息请参考第 6.3.4 节。

6.5 在 4 线模式下传输串行数据（仅UART0）

在 4 线模式下，UART0 使用信号 RTS 和 CTS，以及 RxD 和 TxD 来执行流控制（见第 6.2.2 节）。流控制可以由应用手动或自动执行。下面先分开描述手动流控制的发送和接收，再描述自动流控制。

注 1:

4 线模式是 UART0 的默认模式。只要 vAHI_UartEnable( )被调用，UART 就自动控制 DIO，使之用于 RTS 和 CTS。

注 2:

UART1 没有 4 线模式。

6.5.1 发送数据（4 线模式，手动流控制）

在流控制协议中，只有当目标设备准备好接收时，源设备才发送数据（见第 6.5.2 节）。目标设备准备好接收数据通过激活源设备的 CTS 线来指示。CTS 线的状态可以采样下面任何一种方式来监测：

- 源设备可以使用函数 **u8AHI_UartReadModemStatus( )**来检查 CTS 线的状态；
- 当 CTS 线的状态发生改变时可以产生一个中断；这个中断使用函数 **vAHI_UartSetInterrupt( )**来使能。

一旦检测到 CTS 线的改变，就可以调用下面其中一个函数：

- **vAHI_UartWriteData( )**：这个函数可以用来写单个字节的数据到发送 FIFO。一旦使用，这个函数可以多次调用将数据字节排队进行发送；

- **u16AHI_UartBlockReadData( )**：这个函数可以用来从接收 FIFO 中读出一个数据字节块。函数返回已经成功写入 FIFO 中的字节数。

在 FIFO 中，只要一个数据字节到达 FIFO 头部（假设 TxD 线空闲）就开始发送这个数据。FIFO 到 TxD 线的数据传输由 DMA 引擎自动处理。

请注意，在调用 **vAHI_UartWriteData( )**写数据到发送 FIFO 之前，应用可以使用函数 **u16AHI_UartReadTxFifoLevel( )**或 **u8AHI_UartReadLineStatus( )**检查 FIFO 中是否已经有数据了（前面的传输留下来的）。

当目标设备的 RTS 线变成无效时 CTS 线就失效，见第 6.5.2 节。

6.5.2 接收数据（4 线模式，手动流控制）

在流控制协议中，目标设备只在准备好时才接收数据。当它的接收 FIFO 有足够的空闲空间接收更多数据时就是最常见的情况。应用可以使用函数 **u16AHI_UartReadRxFifoLevel( )**或 **u8AHI_UartReadLineStatus( )**来检查目标设备接收 FIFO 的填充状态。

一旦目标设备上的应用确定它已经准备好接收数据，它就必须通过激活 RTS 线（激活源设备的 CTS 线，见第 6.5.1 节）向源设备请求数据。RTS 线可以使用函数 **vAHI_UartSetRTS( )**或 **vAHI_UartSetControl( )**来激活。

然后，源设备就发送数据，数据从目标设备的 RxD 接收到。RxD 线到接收 FIFO 的本地数据传输由 DMA 引擎自动处理。接收到的数据可以使用下面其中一个函数从接收 FIFO 中读出：

- **u8AHI_UartReadData( )**：这个函数用来从接收 FIFO 读出一个字节的数据；
- **u16AHI_UartBlockReadData( )**：这个函数用来从接收 FIFO 读出一个数据字节块。

之后，应用可以使用函数 **vAHI_UartSetRTS( )**或 **vAHI_UartSetControl( )**使 RTS 线失效来决定停止源设备的传输。这个决定基于接收 FIFO 的填充级别；当 FIFO 中的数据量达到某个级别时，应用开始读取数据，如果应用读取 FIFO 不够快来阻止溢出，它也可以停止传输。接收 FIFO 当前的填充级别可以使用下面任何一种方法来监测：

- 可以调用函数 **u16AHI_UartReadRxFifoLevel( )**来检查接收 FIFO 中当前的数据字节数；
- 当接收 FIFO 中的数据字节数据上升到某个级别时可以产生一个‘接收数据可用’中断；这个中断使用函数 **vAHI_UartSetInterrupt( )**来使能，在这个函数中，中断的触发级别必须设定为 1、4、8 或 14 个字节。

注：

当‘接收数据可用’中断使能时（见上面的描述），接收 FIFO 的‘超时’中断也使能。关于这个中断的更多信息请参考第 6.3.4 节。

6.5.3 自动流控制（4 线模式）

流控制可以在 UART0 的 4 线模式下自动执行，不需要手动执行（如第 6.5.1 节和第 6.5.2 节所述）。自动流控制可以用在目标设备和/或源设备上：

- 在目标设备上，自动流控制不再需要应用监测接收 FIFO 填充级别和激活/撤销 RTS 线；
- 在源设备上，自动流控制不再需要应用在发送数据之前监测 CTS 线。

自动流控制使用函数 **vAHI_UartSetAutoFlowCtrl( )**来配置和使能，如果使用这个函数，函数必须在 UART 使能之后以及数据传输启动之前调用。

vAHI_UartSetAutoFlowCtrl( )函数允许执行下列操作：

- 设定目标设备接收 FIFO 的触发级别（设定成 8、11、13 或 15 个字节），使得：

- 当填充级别低于触发级别时触发本地 RTS 线，指示目标设备准备好接收更多数据；
- 当填充级别处于或高于触发级别时撤销 RTS 线，指示目标设备不适合接收更多数据。

这样，随着目标接收 FIFO 填充级别的上升和下降（随着数据的接收和读取），本地 RTS 线自动应对来控制源设备后面到达的数据。

- 自动监测源设备 CTS 线的使能；当 CTS 线激活时，发送 FIFO 中的所有数据自动发送。

这个函数还允许 RTS/CTS 信号配置成高有效或低有效。

自动流控制可以设置成在两个设备的一个方向或双向数据传输之间执行。

尽管许多数据传输是自动的，源设备的应用程序仍然要写数据到发送 FIFO 中；目标设备的应用程序仍然必须从接收 FIFO 中读取数据。这些操作在下面描述。

发送数据

发送应用程序可以使用下面其中一个函数来发送数据：

- **vAHI_UartWriteData( )**：这个函数可以用来写单个数据字节到发送 FIFO。一旦使用，这个函数可以多次调用来排队数据字节进行发送；
- **u16AHI_UartBlockReadData( )**：这个函数可以用来从接收 FIFO 读取一个数据字节块。函数将返回已经成功写入 FIFO 的字节数。

在 FIFO 中，只要 CTS 线指示目标设备已经准备好接收，数据就通过 TxD 线自动发送。FIFO 到 TxD 线的数据传输由 DMA 引擎自动处理。

请注意，在调用 **vAHI_UartWriteData( )**或 **u16AHI_UartBlockReadData( )**写数据到发送 FIFO 之前，应用程序可以使用函数 **u8AHI_UartReadTxFifoLevel( )**或 **u8AHI_UartReadLine Status( )**来检查 FIFO 中是否已经有数据了（前面的传输遗留下来的）。

接收数据

数据通过目标设备的 RxD 来接收。RxD 线到接收 FIFO 的本地数据传输由 DMA 引擎自动处理。接收到的数据可以使用下面其中一个函数从接收 FIFO 读出：

- **u8AHI_UartReadData( )**：这个函数可以用来从接收 FIFO 读出一个数据字节；
- **u16AHI_UartBlockReadData( )**:这个函数可以用来从接收 FIFO 读出一个数据字节块。

根据下面任何一种方法，应用可以确定何时启动和停止接收 FIFO 数据的读取：

- 可以调用函数 **u16AHI_UartReadRxFifoLevel( )**来检查当前接收 FIFO 中的字符数。这样，当 FIFO 填充级别处于或高于某个阈值时应用程序可以决定开始读取数据；当 FIFO 填充级别处于或低于另一个阈值，或 FIFO 为空时，应用程序可以决定停止读取数据；
- 当接收 FIFO 包含某个数量的数据字节时产生一个‘接收数据可用’中断；这个中断使用函数 **vAHI_UartSetInterrupt( )**来使能，在这个函数中，中断的触发级别必须设定成 1、4、8 或 14 个字节。这样，当这个中断出现时应用可以决定开始从接收 FIFO 读取数据；当 FIFO 中所有接收到的字节已经全部提取出来时应用可以决定停止从接收 FIFO 读取数据。

注:

当‘接收数据可用’中断使能时（见上面的描述），接收 FIFO 的‘超时’中断也使能。关于这个中断的更多信息请参考第 6.3.4 节。

6.6 在 1 线模式下发送串行数据（仅UART1）

在 1 线模式下，UART1 只使用 TxD 线，并且只能发送串行数据。数据在发送设备方便时才发送（因此不执行流控制）。

在这个模式下，UART 使用 **bAHI_UartEnable( )** 使能之前，必须调用函数 **vAHI_UartTxOnly( )**来释放用作 RxD 引脚的控制。既然 RxD 线不使用，也就不需要接收 FIFO 缓冲区，建议用户在 **bAHI_UartEnable( )**中将 RAM 内这个缓冲区的指针设置成 NULL，这样来避免给 FIFO 缓冲区分配 RAM 空间。

6.7 间断条件

在数据传输过程中，如果源设备的应用程序察觉到一个错误，它可以使用函数 **vAHI_UartSetBreak( )**通过设置一个间断条件把这个错误状态告知目标设备。当间断条件发出时，当前正在发送的数据字节被破坏，传输停止。

如果一个 JN516x 器件作为目标设备接收到一个间断条件，就会导致产生一个‘接收线状态’中断（E_AHI_UART_INT_RXLINE），前提是器件的 UART 中断使能。关于 UART 中断的描述请见第 6.3.4 节，关于 UART 中断处理的描述请见第 6.8 节。

函数 **vAHI_UartSetBreak( )**还可以用来清除源设备的间断条件。这时，为了传输发送 FIFO 中留下来的数据，要重新启动发送。

6.8 UART中断处理

在控制 UART 操作的过程中，中断以各种方式被使用。关于 UART 中断各种用法的介绍请参考第 6.3.4 节，与数据传输有关的 UART 中断的使用请参考第 6.4 节和第 6.5 节。

UART 中断由一个用户定义的回调函数来处理，回调函数必须使用 **vAHI_Uart0RegisterCallback( )**或 **vAHI_Uart1RegisterCallback( )**来注册，由具体的 UART（0 或 1）决定。当一个类型 E_AHI_DEVICE_UART0（UART 0）或 E_AHI_DEVICE_UART1（UART 1）的中断出现时，相关的回调函数会自动调用。关于回调函数原型的详细信息请参考附录 A.1。

警告:

已注册的回调函数只在 RAM 保持供电的睡眠模式期间保留。如果 RAM 在睡眠过程中断电而又需要中断，回调函数必须在唤醒调用 **u32AHI_Init( )**之前重新注册。

UART 中断的确切特性（请见第 6.3.4 节的描述）可以从一个传递给回调函数的枚举中识别出来。关于这些枚举的详细信息请参考附录 B.2。

请注意，UART 中断的处理与其他中断的处理方法不同，具体如下：

- 通常，中断的确切原因都是通过一个位图表的方式通知回调函数，但是如果是 UART 中断，使用一个枚举来指示 UART 中断的特性。报告的枚举与当前最高优先级的有效中断条件相对应；
- 通常，中断先自动清除再调用回调函数，但 UART 是特例。当产生一个‘接收数据可用’或‘超时’中断时，UART 中断只在数据从接收 FIFO 中读出时清除。因此，回调函数必须先通过读取接收 FIFO 来处理 UART 的‘接收数据可用’和‘超时’中断，然后再返回。

注:

如果应用队列 API 正在使用，上面关于 UART 中断的问题就由这个 API 来处理，不需要应用程序来处理。关于这个 API 的更多信息请参考*应用队列 API 参考手册（JN-RM-2025）*。

第7章 定时器

本章阐述了使用集成外设 API 函数来控制片上定时器。JN516x 器件有 5 个定时器：定时器 0、定时器 1、定时器 2、定时器 3 和定时器 4。

注：

这些定时器与第 8 章中描述的唤醒定时器和第 9 章中描述的节拍定时器有所不同。

这些定时器提供了下面一系列的操作模式：

- 定时器模式；
- 脉宽调制（PWM）模式；
- 计数器模式；
- 捕获模式；
- Delta-Sigma 模式。

但是，并非全部定时器都可以在所有模式下操作。定时器 1-4 不支持需要外部输入的模式（这些是计数器模式和捕获模式）。定时器模式在第 7.1 节中列出。

如需在上述的其中一种模式下使用定时器，执行下面的操作：

1）首先参考第 7.2 节关于定时器的设置。

2）接着参考第 7.3 节关于定时器的操作（参考所选操作模式对应的小节）。

有关定时器中断的信息，请参考第 7.4 节。

7.1 定时器操作的模式

JN516x 微控制器可使用下面的定时器模式：定时器、脉宽调制（PWM）、计数器、捕获和 Delta-Sigma。下表对这些模式以及每个模式所需的函数（在调用 **vAHI_TimerEnable( )**后）进行总结。除非特别说明，所有 JN516x 定时器都支持这些模式。

表 7.1 定时器操作模式

模式	描述	函数
定时器	源时钟用于产生一个脉冲周期，这个周期由在正脉冲边沿以前的时钟周期数和负脉冲边沿以前的时钟周期数来定义。中断可以在任一边沿或两个边沿上产生。脉冲周期在‘单次触发’模式下只产生一次，在‘重复’模式下连续产生。定时器模式在第 7.3.1 节中进一步说明	**vAHI_TimerConfigureOutputs( )** **vAHI_TimerStartSingleShot( )** 或 **vAHI_TimerStartRepeat( )**
PWM	PWM 模式与定时器模式相同，但在 DIO 管脚上输出的脉宽调制信号除外（这取决于所使用的特定定时器，见第 7.2.1 节）。PWM 模式在第 7.3.1 节中进一步说明	**vAHI_TimerConfigureOutputs( )** **vAHI_TimerStartSingleShot( )** 或 **vAHI_TimerStartRepeat( )**
计数器	定时器用于对作为外部时钟输入的外部输入信号边沿进行计数。定时器可以只计数上升沿或计数上升沿和下降沿。计数器模式在第 7.3.4 节中进一步说明 **定时器 0 支持该模式，但定时器 1-4 不支持该模式**	**vAHI_TimerClockSelect( )** **vAHI_TimerConfigureInputs( )** **vAHI_TimerStartSingleShot( )** 或 **vAHI_TimerStartRepeat( )** **u16AHI_TimerReadCount( )**

续上表

模式	描述	函数
捕获	外部输入信号在源时钟的每个节拍上采样。捕获的结果允许计算采样信号的周期和脉宽。如果需要，可以读取结果而无需停止定时器。捕获模式在第 7.3.3 节中进一步说明 **定时器 0 支持该模式，但定时器 1-4 不支持该模式**	**vAHI_TimerConfigureInputs( )** **vAHI_TimerStartCapture( )** **vAHI_TimerReadCapture( )** 或 **vAHI_TimerReadCaptureFreeRunning( )**
Delta-Sigma	定时器用作一个低速率的 DAC。转换信号在 DIO 管脚上输出（这取决于使用的特定定时器，见第 7.2.1 节），需要简单过滤来得到模拟信号。Delta-Sigma 模式可以在 NRZ 和 RTZ 这两种选择下使用，该模式在第 7.3.2 节中进一步说明	**vAHI_TimerStartDeltaSigma( )**

7.2 设置定时器

本节描述了在定时器启动前如何使用集成外设 API 函数来设置一个定时器（在第 7.3 节中阐述了启动和操作定时器）。

7.2.1 选择DIO

定时器可使用特定的 DIO 管脚，如下表所示。

表 7.2 JN516x 定时器的 DIO 使用

定时器 0 DIO[1]	定时器 1 DIO[1]	定时器 2 DIO[1]	定时器 3 DIO[1]	定时器 4 DIO[1]	功能
8	不使用[2]	不使用[2]	不使用[2]	不使用[2]	时钟或门控输入（在计数器模式下使用）
9	不使用[2]	不使用[2]	不使用[2]	不使用[2]	捕获模式输入
10	11	12	13	17	PWM 和 Delta-Sigma 模式输出

[1] **vAHI_TimerSetLocation( )**可用于将定时器 0 信号从 DIO8-10 传输到 DIO2-4 或将定时器 1-4 信号从 DIO11-13 和 DIO17 传输到 DIO5-8。或者这个函数也可以用于将定时器 2 和定时器 3 信号分别放置于 DO0 和 DO1（数字输出）。

[2] 定时器 1-4 没有输入。

默认情况下，与使能定时器相关的 DIO 管脚保留给定时器使用，但在定时器禁能时变为给通用输入/输出（GPIO）使用。通过调用函数 **vAHI_TimerDIOControl( )**，分配给定时器的 DIO 管脚也可以释放给 GPIO 使用。或者，函数 **vAHI_TimerFineGrainDIOControl( )**也可以将 DIO 配置为同时给所有定时器使用，还可以单独使用这个函数来释放与定时器 0 相关的 3 个 DIO 管脚。

注：

在使用 **vAHI_TimerEnable( )**使能定时器前执行上述的 DIO 配置，以消除定时器操作期间在 GPIO 上产生的干扰。

7.2.2 使能定时器

在启动定时器之前，必须使用函数 **vAHI_TimerEnable( )**来配置和使能定时器。

注：

您必须在尝试其它任何操作之前使能定时器，否则可能会出现异常。

vAHI_TimerEnable( )函数包含了某些配置参数，列举如下：

- 时钟分频器：

为了获得定时器频率，外设时钟被因数 $2^{prescale}$ 分频，其中 *prescale* 是用户可配置的从 0 至 16 的整数值(需要注意的是，值为 0 时钟频率不改变)。例如，在外设时钟频率为 16MHz、*prescale* 的值为 3，分频因子为 8 的情况下，得到的定时器频率为 2MHz。外部晶体振荡器的系统时钟源将给出最稳定的定时器频率（有关系统时钟选项，请参考第 3.1 节）。

- 中断：

每个定时器可以在出现下面的其中一种情况或两种情况时配置为产生中断：

  - 在定时器输出的上升沿（在低电平周期的末尾）；
  - 在定时器输出的下降沿（在整个定时器周期的末尾）。

定时器中断在第 7.4 节中进一步说明。

- 外部输出：

定时器信号可以外部输出，但这个输出必须被明确使能。该输出要求用于 Delta-Sigma 模式和 PWM 模式。这用来区分定时器模式（输出禁能）和 PWM 模式（输出使能）。输出定时器信号的 DIO 管脚取决于设备类型：

  - 对于定时器 0，使用 DIO10；
  - 对于定时器 1，使用 DIO11；
  - 对于定时器 2，使用 DIO12；
  - 对于定时器 3，使用 DIO13；
  - 对于定时器 4，使用 DIO17；

一旦定时器已使用 **vAHI_TimerEnable( )**使能，就可以选择外部时钟输入（如需详情，见第 7.2.3 节），接着使用相关的启动函数就可在所需模式下启动定时器(见第 7.3.1 节到 7.3.4 节)。

注:

使用函数 **vAHI_TimerDisable( )**可以禁能已使能的定时器。这会停止定时器（如果运行）并关断定时器块－这个操作在不需要定时器时很有用，可以减少功耗。应用必须不能尝试访问已禁能的定时器，否则可能会出现异常。

7.2.3 选择时钟

每个定时器都需要一个源时钟，这个时钟由外设时钟来提供。这个源时钟被分频产生定时器的时钟。分频因数在定时器使用 **vAHI_TimerEnable( )**使能时被指定，见第 7.2.2 节。外部晶体振荡器的系统时钟源将给出最稳定的定时器频率（有关系统时钟选项，请参考第 3.1 节）。

当定时器 0 在计数器模式下操作时（见第 7.3.4 节），通过定时器来监控外部时钟。这个信号在 DIO8 管脚上输入，且该输入必须使用函数 **vAHI_TimerClockSelect( )**来使能，这个函数在 **vAHI_TimerEnable( )**之后被调用。

7.3 启动和操作定时器

本节描述了如何使用集成外设 API 函数来启动和操作已设置的定时器，定时器的设置在第 7.2 节中描述。定时器可以在下列模式下启动：

- 定时器或 PWM 模式，见第 7.3.1 节；
- Delta-Sigma 模式，见第 7.3.2 节；
- 捕获模式（仅为定时器 0），见第 7.3.3 节；
- 计数器模式（仅为定时器 0），见第 7.3.4 节。

7.3.1 定时器和PWM模式

定时器模式允许定时器产生一个指定周期的矩形波形，其中这个波形开始为低电平，接着在特定的时间后变为高电平。这些时间在定时器启动时被指定（如下所示），见下面的参数：

- **Time to rise（*μ16Hi*）**：这是启动定时器和（首次）低-到-高跳变之间的时钟周期数。中断可以在这个跳变时产生；
- **Time to fall（*μ16Lo*）**：这是启动定时器和（首次）高-到-低跳变之间的时钟周期数（一个有效的脉冲周期）。中断可以在这个跳变时产生。

这些时间和定时器信号在下面的图 7.1中说明。

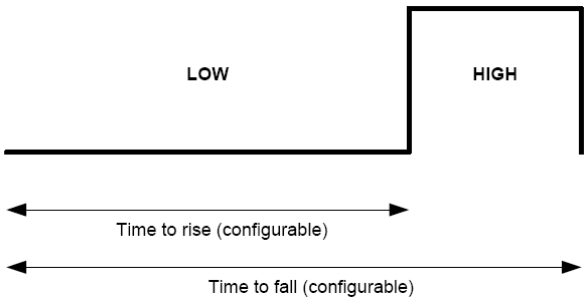

图 7.1 定时器模式信号

在定时器模式内有 2 个子模式，定时器使用不同的函数在这些模式下启动：

- **单次触发模式**：定时器产生一个脉冲周期（如图 7.1所示），接着停止。可使用函数 **vAHI_TimerStartSingleShot( )**在该模式下启动定时器；
- **重复模式**：定时器产生一连串脉冲（其中的重复率由配置的“time to fall”周期来决定，见上面的描述）。可使用函数 **vAHI_TimerStartRepeat( )**在该模式下启动定时器。

一旦启动，就可以使用函数 **vAHI_TimerStop( )**来停止定时器。

PWM（脉宽调制）模式与定时器模式相同，但在 DIO 管脚上输出产生的波形除外，相关的 DIO 请见第 7.2.1 节。这个输出可以在函数 **vAHI_TimerEnable( )**中使能。这个输出也可以使用函数 **vAHI_TimerConfigureOutputs( )**翻转。

7.3.2 Delta-Sigma模式（NRZ和RTZ）

Delta-Sigma模式允许定时器用作一个简单的低速率DAC。这要求在DIO管脚上输出的定时器在调用**vAHI_TimerEnable( )**时使能，相关的DIO见第 7.2.1 节。在这个管脚和地之间必须插入RC（电阻-电容）电路（见图 7.2）。

定时器可使用函数 **vAHI_TimerStartDeltaSigma( )**在 Delta-Sigma 模式下启动。要转换的值通过定时器数字编码为一个伪随机波形，其中：

- 一个波形周期的时钟周期总数是固定的（在 NRZ 情况下是 2^{16}，在 RTZ 情况下是 2^{17}，见下面的描述）；
- 一个周期的高电平时钟周期数设为一个数值，这个值与要转换的值成比例关系；
- 高电平时钟周期在整个周期是随机分布的。

因此，电容将根据指定的值进行充电，这样在周期末尾，产生的电压是数字值的模拟电压。输出电压要求校准，例如，通过测量在高电平周期设为整个脉冲周期（少于一个时钟周期）的变换后电容两端的电压，可以确定可能的最大电压值。

可以使用两种 Delta-Sigma 模式，NRZ 和 RTZ：

- **NRZ（不归零）**：Delta-Sigma NRZ模式使用 16MHz外设时钟，波形周期固定为 2^{16} 个时钟周期。NRZ意味着时钟周期的执行没有间隔（见下面的RTZ选项）。您必须定义在脉冲周期期间高电平状态下的时钟周期数，使得这个高电平周期与要转换的值成比例关系。这个数值在使用函数**vAHI_TimerStartDeltaSigma( )**启动定时器时设置。例如，如果您想转换的值在 0-100 范围内，那么 2^{16} 个时钟周期会对应 100，如需转换的值为 25，您必须将高电平时钟周期数设为 2^{14}（脉冲周期的四分之一）。详情请参考图 7.2；
- **RTZ（归零）**：Delta-Sigma RTZ 模式类似于上述的 NRZ 选项，但在每个时钟周期后会插入一个空（低电平）的时钟周期。因此，每个脉冲周期占用了多个时钟周期的两倍，也就是 2^{17}。需要注意的是，这不会影响正在转换的数值所对应的高电平时钟周期数。如果输出的上升时间和下降时间不同，则这个模式会使转换周期加倍，但却改善了线性度。

注：

有关‘Delta-Sigma’模式的详细信息，请参考微控制器的数据手册。

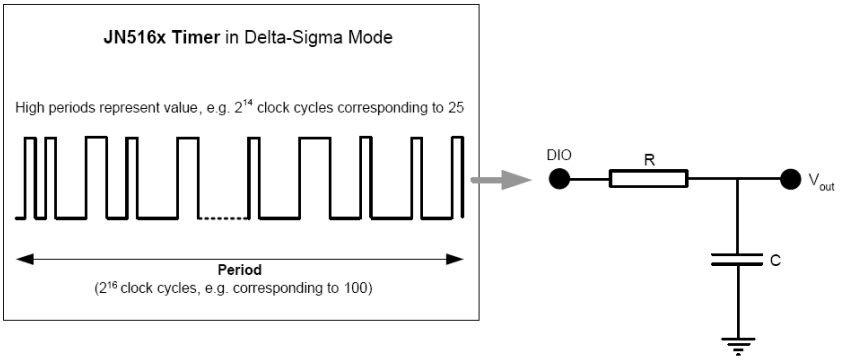

图 7.2 Delta-Sigma NRZ 模式操作

7.3.3 捕获模式

捕获模式仅可用于定时器 0（不可用于定时器 1-4）。在该模式下，定时器可用来测量外部输入的脉宽。外部信号必须在DIO管脚上提供，相关的DIO见 7.2.1 节。从捕获开始到下一个低-到-高跳变以及再到下一个高-到低跳变期间，定时器测量输入信号的时钟周期数。因此，最后脉冲的时钟周期数在这些测量值之间是不同的（见图 7.3）。下面给出了以时间为单位的脉宽：

脉宽（以时间为单位）＝脉冲的时钟周期数×时钟周期

使用函数 **vAHI_TimerStartCapture( )** 在捕获模式下启动定时器。使用函数 **vAHI_TimerReadCapture( )** 可以停止定时器和获得最新的测量值。也可以通过调用 **vAHI_TimerReadCaptureFreeRunning( )**来获得这些测量值，而无需停止定时器。

注：

当调用上述的‘读捕获’函数时，只存储最后一次低-到-高跳变和高-到-低跳变的测量结果，接着再返回。因此，在一个脉冲期间不调用这些函数是很重要的，因为在这种情况下测量将不会得到可靠的结果。为了确保您在一个脉冲结束后获得捕获结果，在使用 **vAHI_TimerEnable( )**配置定时器时应在下降沿使能中断。

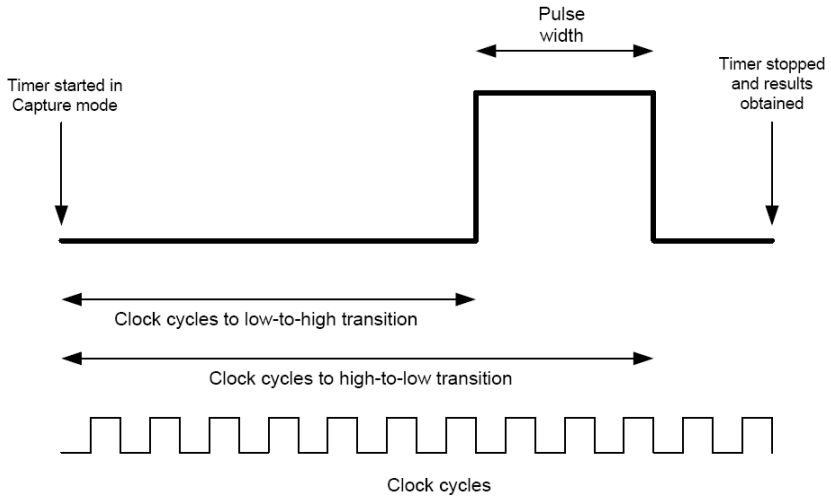

图 7.3 捕获模式操作

捕获模式的输入信号可以被翻转。使用函数 **vAHI_TimerConfigureInputs( )**配置这个选项，并允许测量输入信号的低脉宽（而不是高脉宽）。

7.3.4 计数器模式

计数器模式仅可用于定时器 0（不可用于定时器 1-4）。在该模式下，定时器对外部时钟信号的边沿计数，该信号必须在 DIO 管脚上提供，相关的 DIO 见第 7.2.1 节。通过调用 **vAHI_TimerClockSelect( )**选择一个外部时钟输入来使能计数器模式。

定时器可以只计数上升沿或计数上升沿和下降沿。这个操作必须使用函数 **vAHI_TimerConfigureInputs( )**进行配置。边沿必须至少相隔 100ns，也就是说，脉冲必须宽于 100ns。

与定时器/PWM 模式类似，定时器可在两种子模式的其中一种模式下启动：

- **单次触发模式**：定时器可使用函数 **vAHI_TimerStartSingleShot( )**在该模式下启动，并将在指定的计数值（μ16Lo）处停止；
- **重复模式**：定时器可使用函数 **vAHI_TimerStartRepeat( )**在该模式下启动。定时器连续操作，每次到达指定的计数值（μ16Lo）时计数器复位为 0。

上述的启动函数允许在产生中断时指定两个计数（定时器中断也必须在使用函数 **vAHI_TimerEnable( )**时使能）。

使用函数 **u16AHI_TimerReadCount( )**可随时获取一个正在运行的定时器的当前计数。定时器可以使用函数 **vAHI_TimerStop( )**来停止。

7.4 定时器中断

在函数 **vAHI_TimerEnable( )**中，定时器可以在出现下面的一种情况或两种情况时配置为产生中断：

- 在定时器输出的上升沿（低电平周期的末尾）；
- 在定时器输出的下降沿（整个定时器周期的末尾）。

定时器中断的处理必须包含在特殊定时器用户定义的回调函数中。这些回调函数使用各个定时器专用的注册函数来注册：

- **vAHI_Timer0RegisterCallback( )** 用于定时器 0；

- **vAHI_Timer1RegisterCallback( )** 用于定时器 1；
- **vAHI_Timer2RegisterCallback( )** 用于定时器 2；
- **vAHI_Timer3RegisterCallback( )** 用于定时器 3；
- **vAHI_Timer4RegisterCallback( )** 用于定时器 4。

当出现 E_AHI_DEVICE_TIMER0、E_AHI_DEVICE_TIMER1、E_AHI_DEVICE_TIMER2、E_AHI_DEVICE_TIMER3 或 E_AHI_DEVICE_TIMER4 类型的中断时，会自动调用相关的回调函数。从传递给函数的位图表中可以识别中断的确切信息（上面列出的两种情况的中断）。需要注意的是，在调用回调函数之前中断将自动清除。

注:

回调函数原型详见附录 A.1。中断源信息见附录 B。

警告:

已注册的回调函数只在 RAM 保持供电的睡眠模式期间保留。如果在睡眠过程中 RAM 停止供电并且需要中断，必须先注册回调函数，然后再在唤醒时调用 u32AHI_Init( )。

第8章 唤醒定时器

本章阐述了使用集成外设 API 函数来控制片上唤醒定时器。

JN516x 微控制器包括 2 个唤醒定时器：唤醒定时器 0 和唤醒定时器 1，其中每个定时器是 41-位的计数器。这些唤醒定时器基于 32KHz 时钟（该时钟可源自内部或外部，如 3.1.4 节所述），当器件在睡眠模式下时定时器也可以运行。这些定时器通常用来计时睡眠的持续时间，并且在睡眠周期结束时唤醒器件。唤醒定时器从一个编程的值开始递减计数，并在计数到达 0 时通过产生一个中断或唤醒事件来唤醒器件。

8.1 使用唤醒定时器

本节阐述了如何使用集成外设 API 函数来操作唤醒定时器。

8.1.1 使能和启动唤醒定时器

唤醒定时器使用函数 **vAHI_WakeTimerEnable( )**来使能。这个函数允许使能/禁能在计数器到达 0 时产生的中断。需要注意的是，通过使用函数 **vAHI_SysCtrlRegisterCallback( )**注册的回调函数来处理唤醒定时器中断，见第 3.5 节。

使用函数 **vAHI_WakeTimerStartLarge( )**可以启动唤醒定时器。这个函数将递减计数的起始值作为一个参数，该值必须在 32KHz 的时钟周期内指定（因此，32 对应 1 毫秒）。

在到达 0 时，定时器‘启动’，翻转到 0x1FFFFFFFFF，并继续递减。如果使能，在到达 0 时产生唤醒定时器中断。

注:

如果 32KHz 时钟的时钟源是（默认）内部 32KHz RC 振荡器，那么唤醒定时器的运行速度可能快 18%或慢 18％。为了得到更精确的定时，建议您先校准时钟，并相应调整指定的计数值，如 8.2 节所述。

8.1.2 停止唤醒定时器

使用函数 **vAHI_WakeTimerStop( )** 可随时停止唤醒定时器。计数器将保留定时器停止时的值，并且不会产生中断。

8.1.3 读唤醒定时器

使用函数 **u64AHI_WakeTimerReadLarge( )** 可获取唤醒定时器的当前计数。这个函数不停止唤醒定时器。

8.1.4 获取唤醒定时器状态

使用下列函数可获取唤醒定时器的状态：

- **u8AHI_WakeTimerStatus( )**可用于查找当前正在运行的唤醒定时器；
- **u8AHI_WakeTimerFiredStatus( )**可用于查找已启动的唤醒定时器（传送 0）。唤醒定时器的‘启动’状态也通过这个函数来清除。

注:

1. 如果使用 **u8AHI_WakeTimerFiredStatus( )**来检查唤醒定时器是否产生了一个唤醒事件，那么您必须在 u32AHI_Init()之前先调用这个函数。

2. 如果使用 JenNet 协议，那么从睡眠中唤醒时不要调用 **u8AHI_WakeTimerFiredStatus( )**来获取唤醒定时器中断状态。唤醒时，JenNet 内部调用 **u32AHI_Init( )** 并清除中断状态，然后再将控制传递给应用。若有需要，必须使用系统控制器回调函数来获取中断状态。

8.2 时钟校准

唤醒定时器由 JN516x 微控制器的 32KHz 时钟进行驱动。如果时钟源是内部的 32KHz RC 振荡器，那么定时器的运行速度可能快 18%或慢 18%，这取决于温度、电源电压和厂商容限。为了在这种情况下实现更精确的定时，应使用自校准功能，将 32KHz 时钟与更快更精确的外设时钟相比较，该外设时钟在 16MHz 频率下运行，系统时钟源是外部晶体振荡器（有关系统时钟信息，请参考 3.1 节）。这个测试使用唤醒定时器 0 来执行。当用函数 **vAHI_WakeTimerStart( )** 或 **vAHI_WakeTimerStartLarge( )**来启动唤醒定时器时，这个校准结果允许您校准所需的 32KHz 时钟周期数来实现所需的定时器持续时间。

使用函数 **u32AHI_WakeTimerCalibrate( )**来执行校准，如下所述。

1）唤醒定时器 0 必须禁能（若有需要，使用 **vAHI_WakeTimerStop( )**）。

2）唤醒定时器（0 和 1）的状态必须通过调用函数 **u8AHI_WakeTimerFiredStatus( )**来清除。

3）使用 **u32AHI_WakeTimerCalibrate( )**来启动校准。

这使得唤醒定时器 0 开始递减计数内部 32KHz 时钟的 20 个时钟周期。同时，基准计数器使用 16MHz 外设时钟从 0 开始递增计数。

4）当唤醒定时器到达 0 时，**u32AHI_WakeTimerCalibrate( )**返回通过基准计数器注册的 16MHz 时钟周期数，使这个值为 n。

- 如果时钟在 32KHz 频率下运行，则 n=10000；
- 如果时钟的运行频率低于 32KHz，则 n>10000；
- 如果时钟的运行频率高于 32KHz，则 n<10000。

5）接着你可以计算所需的 32KHz 时钟周期数（**vAHI_WakeTimerStart( )** 或 **vAHI_WakeTimerStartLarge( )**）来实现所需的定时器持续时间。如果 T 是所需的持续时间（以秒计算），那么 32KHz 时钟周期数 N 由下式得出：

$$N = \left(\frac{10000}{n}\right) \times 32000 \times T$$

例如，如果 n 得到的一个值为 9000，则意味着 32KHz 时钟正在高速运行。因此，为了实现 2 秒的定时器持续时间，而不是所需的 64000 个时钟周期，您将需要(10000/9000)×32000×2 个时钟周期，也就是 71111（四舍五入）。

提示:

为了确保器件及时唤醒来处理一个预定的事件，最好将所需的 32KHz 时钟周期数估计得低一些而不是将它们估计得高一些。

第9章 节拍定时器

本章阐述了使用集成外设 API 函数来控制节拍定时器。

节拍定时器是一个硬件定时器，它来自外设时钟并且可以用来执行下面的操作：

- 软件的定时中断；
- 常规事件，例如软件定时器或操作系统的节拍；
- 高精度定时参考；
- 系统监控超时，比如在看门狗定时器中使用。

注：

对于高精度的节拍定时器操作，外设时钟应在 16MHz 频率下运行，系统时钟源是外部晶体振荡器。有关系统时钟的信息，请参考第 3.1 节。

9.1 节拍定时器操作

节拍定时器递增计数直至计数匹配一个预定义的参考值（可指定起始值）。定时器可在三种模式的其中一种模式下操作，这确定了一旦到达参考计数时定时器将要执行的操作，如下所示：

- 继续递增计数；
- 从 0 开始重新计数；
- 停止计数（单次触发模式）。

也可以使能在到达参考计数时产生的中断。

9.2 使用节拍定时器

本节描述了如何使用集成外设 API 函数来设置和运行节拍定时器。

9.2.1 设置节拍定时器

在器件上电/复位时，禁能节拍定时器。但是，在设置节拍定时器之前，建议您调用函数 **vAHI_TickTimerConfigure( )**并指定禁能的选项。接着可以如下设置起始计数和参考计数：

1）使用函数 **vAHI_TickTimerWrite( )**来设置起始计数（范围从 0 到 0xFFFFFFFF）。需要注意的是，如果在定时器使能时调用这个函数，则定时器将立即从指定的值开始计数。

2）使用函数 **vAHI_TickTimerInterval( )**来设置参考计数（范围从 0 到 0x0FFFFFFF）。

9.2.2 运行节拍定时器

一旦定时器已设置（如 9.2.1 节所述），通过再次调用函数 **vAHI_TickTimerConfigure( )**可启动定时器，但此时应指定第 9.1 节中列出的三种可操作模式的其中一种模式。

通过调用函数 **u32AHI_TickTimerRead( )**可随时获得节拍定时器的当前计数。

需要注意的是，如果节拍定时器在单次触发模式下启动，一旦停止（在到达参考计数时），通过使用 **vAHI_TickTimerWrite( )**设置另一个起始值可再次启动定时器。

9.3 节拍定时器中断

可使能当节拍定时器到达其参考计数时产生的中断。使用函数 **vAHI_TickTimerIntEnable( )** 来使能这个中断。

节拍定时器中断由使用函数 **vAHI_TickTimerRegisterCallback( )**注册的用户定义的回调函数来处理。

当出现 E_AHI_DEVICE_TICK_TIMER 类型的中断时，会自动调用注册的回调函数。有关

回调函数原型的详细内容，请参考附录 A.1。

警告:

已注册的回调函数只在 RAM 保持供电的睡眠模式期间保留。如果在睡眠过程中 RAM 停止供电并且需要中断，必须先注册回调函数，然后再在唤醒时调用 u32AHI_Init( )。

另外，还提供下列函数来处理节拍定时器中断的状态：

- **bAHI_TickTimerIntStatus( )**获取节拍定时器的当前中断状态；
- **vAHI_TickTimerIntPendClr( )**清除等待处理的节拍定时器中断。

第10章 看门狗定时器

本章阐述了使用集成外设 API 函数来控制 JN516x 器件的看门狗定时器。

看门狗定时器可以使 JN516x 器件从软件锁定中恢复。需要注意也可使用在第 9 章中描述的节拍定时器来执行看门狗。

10.1 看门狗操作

看门狗定时器执行一个超时周期，它来自内部高速 RC 振荡器（该振荡器运行在 27MHz 或 32MHz 频率下）。

在到达超时周期时，JN516x 器件自动复位。因此，为了避免芯片复位，应用必须定期复位看门狗定时器（复位到超时周期的起始处），以防止定时器的定时时间已到并指示应用仍控制 JN516x 器件。如果定时器到达定时时，假设应用已不能控制芯片，那么芯片的硬件复位自动启动。

需要注意的是，看门狗定时器在休眠模式下继续运行，但在睡眠模式或深度睡眠模式下不运行，或当硬件调试器已控制 CPU 时也不运行（但是，当调试器重启（un-stall）CPU 时定时器将自动重启）。

注:

1. 在上电、复位或从睡眠模式下唤醒后，看门狗定时器使能，可能的最大超时周期为 16392ms（不管在任何睡眠或复位之前它的状态如何）。

2. 可配置看门狗定时器调用一个超时异常。这允许调试在应用开发期间导致超时的情况。详情请参考第 10.2.3 节。

10.2 使用看门狗定时器

本节描述了如何使用集成外设 API 函数来启动和复位看门狗定时器。

10.2.1 启动定时器

看门狗定时器默认在 JN516x 器件上启动。启动时，可能的最大超时为 16392ms。

- 如果要求看门狗定时器的超时周期短一些，则定时器必须以所希望的周期重启。为了完成这个操作，先调用函数 **vAHI_WatchdogRestart( )**，从超时周期起始处重启定时器，接着调用函数 **vAHI_WatchdogStart( )**来指定新的超时周期（见下面的描述）。
- 如果应用不需要看门狗定时器，则在开始编程时调用函数 **vAHI_WatchdogStop( )**来停止定时器。

在函数 **vAHI_WatchdogStart( )**中，必须通过一个系数 *prescale*（范围从 0 到 12）来指定超时周期，根据下列公式，函数使用 *prescale* 的值来计算超时周期（以毫秒为单位）：

超时周期＝8ms　　如果 *Prescale*=0

超时周期＝$[2^{(Prescale-1)}+1]\times 8ms$　　如果 $1 \leq Prescale \leq 12$

这得出超时周期的范围为 8ms 到 16392ms。

需要注意的是，如果看门狗定时器的时钟源是内部 RC 振荡器，那么由于振荡器的偏差，得到的实际超时周期可能比计算的值少 18%。

注:

如果看门狗定时器在停止状态时调用 **vAHI_WatchdogStart( )**，则以指定的超时周期启动定时器。如果在定时器运行时调用这个函数，则定时器将以新指定的超时周期继续运行。

警告:

确保设置的看门狗超时周期大于最坏情况下的 Flash 存储器读-写周期。如果在 Flash 存储器访问时看门狗超时，则 JN516x 微控制器将进入编程模式。有关读-写周期的信息，请参考相关的 Flash 存储器数据手册。

使用函数 **u16AHI_WatchdogReadValue( )**可获取正在运行的看门狗定时器的当前计数。

10.2.2 复位定时器

一个正在运行的看门狗定时器在到达预先设置的超时周期前应通过应用复位。这个操作使用函数 **vAHI_WatchdogRestart( )**来完成，在超时周期开始时重启定时器。当使用这个复位时，应用要考虑的情况是真正的超时周期可能比计算的超时周期少 18%（如果定时器的时钟源是内部 RC 振荡器，见 10.2.1 节）。

如果应用不能预防看门狗超时，则芯片将自动复位。在芯片复位后可以使用函数 **bAHI_WatchdogResetEvent( )**来找出最后的硬件复位是否由看门狗定时器定时时间已到事件产生。

需要注意的是，通过使用函数 **vAHI_WatchdogStop( )**也可以停止看门狗定时器和冻结它的计数。

10.2.3 用于调试的异常处理程序

默认情况下，看门狗定时器的定时时间已到将会使 JN516x 器件复位。在定时器定时时间已到时也可以调用一个异常。这个异常通过栈溢出异常处理程序来服务，该处理程序可调用函数 **bAHI_WatchdogResetEvent( )**来确定是否出现看门狗异常。这有助于调试导致看门狗超时的情况。因此，这种选择专用于应用开发过程。

通过调用函数 **vAHI_WatchdogException( )**来使能异常。在使能看门狗异常之前，应开发栈溢出异常处理程序函数。

注：栈溢出异常处理程序函数应具有下列的原型定义:

PUBLIC void vException_StackOverflow(void);

我们不期望 C 语言写的异常处理程序返回，一旦已执行任何行动，它应处于循环状态或复位器件。

第11章 脉冲计数器

本章阐述了使用集成外设 API 函数来控制 JN516x 器件的脉冲计数器。

JN516x 器件上有两个脉冲计数器，脉冲计数器 0 和脉冲计数器 1。脉冲计数器检测和计数在相关的 DIO 管脚上输入的外部信号的脉冲。

11.1 脉冲计数器操作

两个脉冲计数器（脉冲计数器 0 和脉冲计数器 1）均为 16-位的计数器，默认情况下分别接收 DIO1 和 DIO8 管脚上的输入信号（脉冲计数器 0 也可以使用 DIO4 的输入，脉冲计数器 1 也可以使用 DIO5 的输入）。若有需要，这两个计数器可组合在一起形成一个 32-位的计数器，在这种情况下，DIO 的输入信号可以从两个计数器的输入管脚中选择。

脉冲计数器可以在 JN516x 器件的所有电源模式下操作（包括睡眠模式），输入信号高达 100KHz。计数器可配置为出现相应输入的上升沿或下降沿时递增。每个脉冲计数器有一个相关的用户定义的参考值。当计数器通过它预先配置的参考值，也就是当计数到达*（参考值+1）*时，计数器可以产生一个中断（或若在睡眠模式下产生唤醒事件）。计数器在到达最大计数值时不会停止，而是循环回 0。

注:

脉冲计数器中断由系统控制器中断的回调函数来处理，这些函数使用 **vAHI_SysCtrlRegisterCallback( )**注册，见 11.3 节。

去抖

输入脉冲可使用 32KHz 时钟去抖，以防止在低速边沿或噪声边沿上进行错误计数。在识别输入信号的变化前，去抖特性需要多次相同的连续输入采样（2、4 或 8）。根据去抖设置，脉冲计数器可按下列频率与输入信号操作：

- 如果去抖禁能，则频率为 100KHz；
- 如果出现 2 次连续采样时去抖使能，则频率为 3.7KHz；
- 如果出现 4 次连续采样时去抖使能，则频率为 2.2KHz；
- 如果出现 8 次连续采样时去抖使能，则频率为 1.2KHz；

当配置脉冲计数器时选择所需的去抖设置，如 11.2.1 节所述。

当使用去抖时必须激活 32KHz 时钟，因此，为了得到最少的睡眠电流，不应使用去抖特性。

11.2 使用脉冲计数器

本节描述了如何使用集成外设 API 函数来配置、启动/停止和监控脉冲计数器。

11.2.1 配置脉冲计数器

必须先使用函数 **bAHI_PulseCounterConfigure( )**来配置脉冲计数器。这个函数调用必须指定：

- 两个 16-位的脉冲计数器是否组合为一个 32-位的脉冲计数器，如果是，则组合计数器上的管脚将使用它的输入；
- 脉冲计数值是否在输入信号一个脉冲的上升沿或下降沿递增；
- 是否使能去抖特性，如果是，将出现连续的采样次数（2、4、8）（见第 11.1 节）；
- 是否使能一个中断，这个中断在脉冲计数通过参考值时产生（见下面的描述）。

当使用这个函数选择脉冲计数器时，将自动使用相应管脚的输入信号：脉冲计数器 0 使用

DIO1，脉冲计数器 1 使用 DIO8（组合脉冲计数器可使用这些 DIO 中任一 DIO 的输入）。但是，使用 **vAHI_PulseCounterSetLocation( )** 可将输入传输到另一个管脚。

- 对于脉冲计数器 0，输入可以从 DIO1 传输到 DIO4；
- 对于脉冲计数器 1，输入可以从 DIO8 传输到 DIO5。

通过调用函数 **bAHI_SetPulseCounterRef( )**来完成脉冲计数器的配置以设置参考计数。需要注意的是，脉冲计数器将继续计数超过指定的参考值，但在到达可能的最大计数值时循环到 0。

11.2.2 启动和停止脉冲计数器

使用函数 **bAHI_StartPulseCounter( )**来启动一个配置的脉冲计数器。需要注意的是，当调用这个函数时，计数值可能加 1（即使没有检测到脉冲）。

脉冲计数器将继续计数直至使用函数 **bAHI_StopPulseCounter( )**停止，此时计数将被冻结。接着计数可使用下面的其中一个函数来清零：

- **bAHI_Clear16BitPulseCounter( )**用于脉冲计数器 0 或 1；
- **bAHI_Clear32BitPulseCounter( )**用于组合脉冲计数器。

11.2.3 监控脉冲计数器

应用可检测一个正在运行的脉冲计数器是否已到达它的参考计数，方式如下：

- 可使能一个中断，这个中断在通过参考计数时被触发（见第 11.3 节）；
- 应用可使用函数 **u32AHI_PulseCounterStatus( )**来轮询脉冲计数器，这个函数返回了一个包含所有运行的脉冲计数器的位图表，并指示每个计数器是否已到达它的参考值。

所提供的函数还可以读取脉冲计数器的当前计数而无需停止脉冲计数器或清除它的计数。所需的函数取决于脉冲计数器：

- **bAHI_Read16BitCounter( )**用于脉冲计数器 0 或 1；
- **bAHI_Read32BitCounter( )**用于组合脉冲计数器。

当脉冲计数器到达它的参考计数时，它继续计数超过这个参考值。若有需要，可使用函数 **bAHI_SetPulseCounterRef( )**来设置新的参考计数（当计数器正在运行时）。

11.3 脉冲计数器中断

当计数器的计数通过预先配置的参考值、也就是当计数到达*（参考值+1）*时，脉冲计数器可以选择产生一个中断。这个中断使能可作为函数 **bAHI_PulseCounterConfigure( )** 调用的一部分。

注：

脉冲计数器在睡眠过程中继续运行。脉冲计数器中断可用于将 JN516x 器件从睡眠模式中唤醒。

脉冲计数器中断作为系统控制器中断来处理，必须包含在用户定义的回调函数中，使用函数 **vAHI_SysCtrlRegisterCallback( )**来注册，见第 3.5 节。

当出现 E_AHI_DEVICE_SYSCTRL 类型的中断时，自动调用注册的回调函数。如果中断源是脉冲计数器 0 或脉冲计数器 1，那么将会在传递给回调函数的位图表中指示（如果使用组合脉冲计数器，则该计数器将在中断出现时作为脉冲计数器 0）。需要注意的是，在调用回调函数之前中断将自动清零。

一旦出现了脉冲计数器中断，脉冲计数器将继续计数超出它的参考值。若有需要，可使用函数 **bAHI_SetPulseCounterRef( )**来设置新的参考计数（当计数器正在运行时）。

第12章 红外发送器

本章阐述了使用集成外设 API 函数来控制 JN516x 器件的红外发送器。

红外发送是定时器 2 的一种特殊特性，它使用定时器产生波形用于红外远程控制应用。

12.1 红外发送器操作

远程控制协议（比如说 Philips RC-6）使用编码的位流将 On-Off Key（OOK）调制应用于载波信号。红外发送器能够适应各种远程控制协议，这些协议具有不同载波频率、载波占空比和数据位编码要求。红外发送器使用定时器 2 来产生一个可编程的载波波形，这个波形通过 RAM 中存储的可编程位序列进行 OOK 调制。最后得到的波形输出到相关的定时器 2 输出管脚。

警告:

一个典型的红外 LED 要求至少 15mA 的驱动电流。由于标准的数字输出不具备这种驱动能力，因此将会要求使用外部晶体管或 LED 驱动器来提供这种电流。

示例波形

OOK调制波形的示例如图 12.1所示。

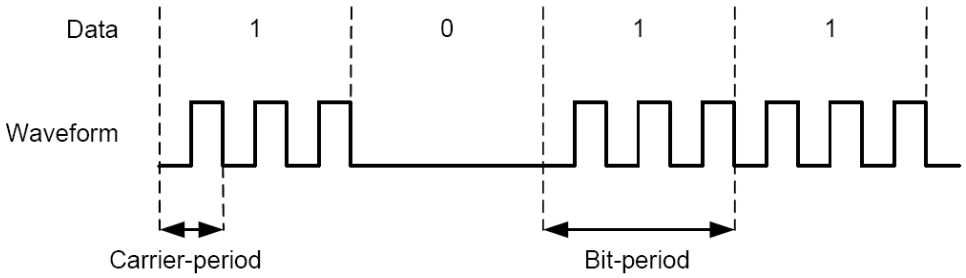

图 12.1 OOK 调制波形示例

在这个示例中，周期性载波信号和二进制位模式 1011 的**逻辑与**产生最终的 OOK 调制波形，其中每个数据位的周期等于载波周期的 3 倍。

12.2 使用红外发送器

本节描述了如何使用集成外设 API 函数来配置、启动和监控红外发送。

注:

当使用定时器 2 的红外发送特性时，不要为定时器 2 调用在第 7 章和第 22 章中列举的公用定时器函数，若有需要，**vAHI_TimerSetLocation( )**和 **vAHI_TimerFineGrainDIOControl( )**除外。

12.2.1 配置红外发送器

红外发送器必须先使用 **bAHI_InfraredEnable( )**函数来使能和配置。这个函数调用必须指定：

- 使用时钟预分频值来递减分频外设时钟并产生定时器时钟；
- 在启动定时器后载波变高之前的定时器时钟周期数，这定义了载波低电平的持续时间；
- 在启动定时器后载波再次变低之前的定时器时钟周期数，这定义了载波周期；
- 以载波周期为单位的位周期；
- 输出信号极性；
- 如果中断使能，则表示发送结束。

示例配置

Philips RC-6 协议需要 36KHz±10%的载波信号，占空比在 25%到 50%之间。在这个示例中，我们将使用 30%的占空比。

RC-6 协议编码逻辑‘0’为位‘01’，逻辑‘1’为位‘10’，前面的符号为位‘11111100’。每个独立位的持续时间是载波周期的 16 倍（即 16×1/36KHz≈444μs）。这些波形的定时要求可通过调用 **bAHI_InfraredEnable( )**来满足，使用下列输入参数值：

- *u8Prescale*: 2 (定时器时钟周期 = $2^{u8Prescale}$/16MHz = 4/16MHz = 250ns)；
- *u16Hi*: 78 (载波低电平持续时间 = 78x250ns = 19.5μs)；
- *u16Lo*: 111 (载波周期 = 111x250ns = 27.75μs, 即，频率 = 36.036kHz)；
- *u16BitPeriodInCarrierPeriods*: 16 (位周期 = 16×27.75μs = 444μs)；
- *bInvertOutput*: TRUE 或 FALSE，根据外部晶体的要求；
- bInterruptEnable: TRUE 或 FALSE，根据应用的要求。

注:

为了保证得到精确的波形定时，建议用户确保外设时钟在 16MHz 频率下操作，系统时钟源是外部晶体振荡器，见 3.1 节。

12.2.2 启动红外发送

通过调用 **bAHI_InfraredEnable( )**为远程控制协议配置波形定时要求，用户可通过调用函数 **bAHI_InfraredStart( )**来启动红外发送的波形发生。这个函数调用必须指定：

- 32-位宽阵列的 RAM 起始地址包含了要发送的编码位（最大的阵列大小为 128 字）；
- 要发送阵列的编码的位的数目（1～4096 位）。

在调用 **bAHI_InfraredStart( )**之前，用户应该以所需的发送编码位模式来安排数据阵列。首先将发送每个 32-位字的 MSB。例如，发送 35 个位将要求用户编程所有的 32 个位，先是 32-位字，后面是第二个 32-位字的高 3 位。

注:

数据阵列应包含一个编码的位序列。应用有责任按照协议的要求执行这种编码。

在调用 **bAHI_InfraredStart( )**时，将启动波形发生，器件将自动使用 DMA 机制从数据阵列中读取指定的位数，并产生一个使用预先配置定时要求的 OOK 调制载波波形。

默认情况下，产生的波形将被输出到管脚 DIO12（即，定时器 2 的默认输出管脚）。若有需要，通过调用函数 **vAHI_TimerSetLocation( )**将定时器 2 输出传输到管脚 DIO6 或管脚 DO0，见第 7.2.1 节。

注:

为了预防在与定时器 2 相关的输出管脚上出现干扰，我们建议应用在 **bAHI_InfraredEnable( )**之前调用 **vAHI_TimerSetLocation( )**。

12.2.3 监控红外发送

应用可通过下面的其中一种方式来检测红外发送何时完成：

- 可使能一个中断，这个中断在发送完成时触发（见 12.3 节）；
- 应用可使用函数 **bAHI_InfraredStatus( )**来轮询红外发送状态，如果正在发送则该函数返回 TRUE，否则返回 FALSE。

12.2.4 禁能红外发送器

如果红外发送器使能，那么通过调用函数 **vAHI_InfraredDisable( )**可以将其禁能。在调用这个函数后，必须先调用 **bAHI_InfraredEnable( )**，再尝试调用其它任何红外函数。

12.3 红外发送器中断

红外发送器可以在红外发送完成时产生一个中断。这个中断使能可作为函数 **bAHI_InfraredEnable( )** 调用的一部分。

这个中断作为红外发送器中断来处理，并且必须包含在用户定义的回调函数中，回调函数用函数 **vAHI_InfraredRegisterCallback( )**来注册。

当出现 E_AHI_DEVICE_INFRARED 类型的中断时，自动调用注册的回调函数。中断源将会在传递给回调函数的位图表中指示。需要注意的是，在调用回调函数之前中断将自动清零。

第13章 串行接口（SI）

本章阐述了使用集成外设 API 函数来控制 2-线串行接口（SI）。

JN516x 微控制器包含了一个业界标准的 2-线同步串行接口，该接口为器件提供了一个简单且高效的数据交换方式。串行接口类似于 I^2C 接口，它包含了两条线：

- DIO15 上的串行数据线；
- DIO14 上的串行时钟线。

这些信号可以分别传输到 DIO17 和 DIO16。

JN516x 器件的 SI 外设可用作串行接口总线的主机或从机：

- SI 主机功能在第 13.1 节中描述；
- SI 从机功能在第 13.2 节中描述。

提示:

串行接口使用的协议在I^2C规范中详述（可访问www.nxp.com网站）。

13.1 SI主机

SI 主机可以与串行接口总线上的从机设备执行任何方向的通信。本节描述了如何执行一个数据传输。

注:

JN516x 器件的串行接口总线可以有多于一个的主机，但多个主机不能同时使用总线。为了避免发生这种情况，器件提供仲裁机制来解决通过竞争主机试图控制串行接口总线所造成的冲突。如果主机丢失仲裁，那么它必须等待并且稍后再尝试。

13.1.1 使能SI主机

SI 主机在集成外设 API 中有它自身的函数集（SI 从机有一个独立的函数集）。在使用任何 SI 主机函数之前，SI 外设必须使用函数 **vAHI_SiMasterConfigure( )**来使能。

使能时，接口将 DIO14 管脚用作时钟线，DIO15 管脚用作双向数据线。但是，这些信号可以使用函数 **vAHI_SiSetLocation( )** 分别传输到 DIO16 和 DIO17。

作为总线主机，微控制器提供时钟（在时钟线上）用于同步数据传输（在数据线上），这个时钟从外设时钟分频得到，外设时钟必须在 16MHz 频率下运行（系统时钟的时钟源必须是外部晶体振荡器，有关系统时钟的信息请参考第 3.1 节）。在接口使能时指定时钟分频因数 *PreScaler*，接口的最后操作频率由下式得到：

操作频率＝16/[(*PreScaler* + 1)×5]MHz

SI 使能函数还允许使能 SI 中断（属于类型 E_AHI_DEVICE_SI），该中断由用户定义的回调函数来处理，回调函数用函数 **vAHI_SiRegisterCallback( )**注册。有关回调函数原型的详细内容，请参考附录 A.1。

警告:

已注册的回调函数只在 RAM 保持供电的睡眠模式期间保留。如果在睡眠过程中 RAM 停止供电并且需要中断，必须先注册回调函数，然后再在唤醒时调用 u32AHI_Init( )。

vAHI_SiMasterConfigure( )还允许脉冲抑制滤波器使能，该滤波器抑制在时钟和数据线上脉宽少于 62.5ns 的任何假脉冲（高或低）。同时要注意的是，使用这个函数使能的 SI 主机稍后可以使用 **vAHI_SiMasterDisable( )**禁能。

13.1.2 写数据到SI从机

下列过程阐述了 SI 主机如何写数据到 7-位或 10-位地址的 SI 从机。假设 SI 主机已使能，如 13.1.1 节所述。数据可包含一个或多个字节。

步骤 1 控制 SI 总线并写从机地址到总线

SI 主机必须先控制 SI 总线，再发送目标从机的地址进行数据传输。7-位从机地址和 10-位从机地址所需的方法有所不同，如下所示：

对于 7-位从机地址：

a) 调用函数 **vAHI_SiMasterWriteSlaveAddr( )**来指定 7-位从机地址，并通过这个函数来指定将在从机上执行写操作。这个函数将把指定的从机地址放到 SI 主机的缓冲区，但不会在 SI 总线上发送。

b) 调用函数 **bAHI_SiMasterSetCmdReg( )**来发布启动命令和写命令，以控制 SI 总线并发送上面指定的从机地址。

c) 等待成功的指示（从机地址发送和目标从机响应），这通过轮询或等待一个中断来实现，有关这个阶段的详细内容，请参考第 13.1.4 节。

对于 10-位从机地址：

a) 调用函数 **vAHI_SiMasterWriteSlaveAddr( )**来指示将要使用的 10-位从机地址，并指定相关从机地址的两个最高位（指定时，这些位必须与 0x78 逐位**相或**）。同时通过这个函数来指定将在从机上执行写操作。这个函数将会把指定的信息放到 SI 主机的缓冲区，但不会在 SI 总线上发送。

b) 调用函数 **bAHI_SiMasterSetCmdReg( )**来发布启动命令和写命令，以控制 SI 总线并发送上面指定的从机地址信息。

c) 等待成功的指示（从机地址发送和至少一个匹配的从机响应），这通过轮询或等待一个中断来实现，有关这个阶段的详细内容，请参考第 13.1.4 节。

d) 调用函数 **vAHI_SiMasterWriteData8( )**来指定从机地址 8 个保留的位。这个函数将会把指定的信息放到 SI 主机的缓冲区，但不会在 SI 总线上发送。

e) 调用函数 **bAHI_SiMasterSetCmdReg( )**来发布一个写命令，以发送上面指定的从机地址信息。

f) 等待成功的指示（从机地址信息发送和目标从机响应），这通过轮询或等待一个中断来实现，有关这个阶段的详细内容，请参考第 13.1.4 节。

步骤 2 发送数据字节到从机

如果只有一个数据字节或最后的数据字节发送到从机，那么直接跳转到步骤 3，否则按下面的指令来执行：

a) 调用函数 **vAHI_SiMasterWriteData8( )**来指定要发送的数据字节。这个函数将会把指定的数据放到 SI 主机的缓冲区，但不会在 SI 总线上发送。

b) 调用函数 **bAHI_SiMasterSetCmdReg( )**来发布写命令，以发送上面指定的数据字节。

c) 等待成功的指示（数据字节发送和目标从机响应），这通过轮询或等待一个中断来实现，有关这个阶段的详细内容，请参考第 13.1.4 节。

对于所有后续的数据字节，重复上面的指令（步骤 2a-c），但最后要发送的字节除外（这个在步骤 3 中说明）。

步骤 3 发送最后的数据字节到从机

如下发送最后的（或一个）数据字节到从机：

a) 调用函数 **vAHI_SiMasterWriteData8( )**来指定要发送的数据字节。这个函数将会把指定的数据放到 SI 主机的缓冲区，但不会在 SI 总线上发送。

b) 调用函数 **bAHI_SiMasterSetCmdReg( )**来发布写命令和停止命令，以发送上面指定的数据字节并释放对 SI 总线的控制。

c) 等待成功的指示（数据字节发送和目标从机响应），这通过轮询或等待一个中断来实现，有关这个阶段的详细内容，请参考第 13.1.4 节。

13.1.3 读取SI从机的数据

下列过程阐述了 SI 主机如何从 7-位或 10-位地址的 SI 从机中读取数据发送。假设 SI 主机已使能，如 13.1.1 节所述。数据可包含一个或多个字节。

步骤 1 控制 SI 总线并写从机地址到总线

SI 主机必须先控制 SI 总线，再发送从机的地址，该从机是数据传输的起点。7-位从机地址和 10-位从机地址要求的方法有所不同，如下所示：

对于 7-位从机地址：

a) 调用函数 **vAHI_SiMasterWriteSlaveAddr( )**来指示 7-位从机地址，并通过这个函数来指定将在从机上执行读操作。这个函数将会把指定的从机地址放到 SI 主机的缓冲区，但不会在 SI 总线上发送。

b) 调用函数 **bAHI_SiMasterSetCmdReg( )**来发布启动命令和写命令，以控制 SI 总线并发送上面指定的从机地址。

c) 等待成功的指示（从机地址发送和目标从机响应），这通过轮询或等待一个中断来实现，有关这个阶段的详细内容，请参考第 13.1.4 节。

对于 10-位从机地址：

a) 调用函数 **vAHI_SiMasterWriteSlaveAddr( )**来指示将要使用的 10-位从机地址，并指定相关从机地址的两个最高位。同时通过这个函数来指定将在从机上执行写操作。这个函数将会把指定的信息放到 SI 主机的缓冲区，但不会在 SI 总线上发送。

b) 调用函数 **bAHI_SiMasterSetCmdReg( )**来发布启动命令和写命令，以控制 SI 总线并发送上面指定的从机地址信息。

c) 等待成功的指示（从机地址信息发送和至少一个匹配的从机响应），这通过轮询或等待一个中断来实现，有关这个阶段的详细内容，请参考第 13.1.4 节。

d) 调用函数 **vAHI_SiMasterWriteData8( )**来指定从机地址 8 个保留的位。这个函数将会把指定的信息放到 SI 主机的缓冲区，但不会在 SI 总线上发送。

e) 调用函数 **bAHI_SiMasterSetCmdReg( )**来发布一个写命令，以发送上面指定的从机地址信息。

f) 等待成功的指示（从机地址信息发送和目标从机响应），这通过轮询或等待一个中断来实现，有关这个阶段的详细内容，请参考第 13.1.4 节。

g) 再次调用函数 **vAHI_SiMasterWriteSlaveAddr( )**来指示将要使用的 10-位从机地址，并指定相关从机地址的两个最高位。这时，通过这个函数来指定将在从机上执行读操作。这个函数将会把指定的信息放到 SI 主机的发送缓冲区，但不会在 SI 总线上发送。

h) 调用函数 **bAHI_SiMasterSetCmdReg( )**来发布启动命令和写命令，以控制 SI 总线并发

送上面指定的从机地址信息。

i) 等待成功的指示，这通过轮询或等待一个中断来实现，有关这个阶段的详细内容，请参考第 13.1.4 节。

步骤 2 读取从机的数据字节

如果仅读取从机的一个数据字节或最后的数据字节，那么直接跳转到步骤 3，否则按下面的指令来执行：

a) 调用函数 **bAHI_SiMasterSetCmdReg( )**来发布一个读命令，以请求从机的一个数据字节。一旦接收到字节，就使用这个函数来使能一个要发送到从机的 ACK（应答）。

b) 等待成功的指示（读请求发送和数据接收），这通过轮询或等待一个中断来实现，有关这个阶段的详细内容，请参考第 13.1.4 节。

c）调用函数 **u8AHI_SiMasterReadData8( )**来读取从 SI 主机缓冲区中接收到的数据字节。

对于所有后续的数据字节，重复上面的指令（步骤 2a-c），但最后要读取的字节除外（这个在步骤 3 中说明）。

步骤 3 读取从机最后的数据字节

如下读取从机的最后（或一个）数据字节：

a) 调用函数 **bAHI_SiMasterSetCmdReg( )**来发布读命令和停止命令，以请求从机的一个数据字节并释放对 SI 总线的控制。一旦接收到字节，就使用这个函数来使能一个要发送到从机的 NACK（来指示不再需要数据）。

b) 等待成功的指示（读请求发送和数据接收），这通过轮询或等待一个中断来实现，有关这个阶段的详细内容，请参考第 13.1.4 节。

c）调用函数 **u8AHI_SiMasterReadData8( )**来读取从 SI 主机缓冲区中接收到的数据字节。

13.1.4 等待完成

对于第 13.1.2 节和第 13.1.3 节写过程和读过程的各小点，在继续操作之前必须等待一个操作成功的指示。应用可使用中断或轮询来确定何时继续：

- **中断**：当调用 **vAHI_SiConfigure( )**或 **vAHI_SiMasterConfigure( )**时，可使能 SI 中断，如第 13.1.1 节所述。SI 中断（属于类型 E_AHI_DEVICE_SI）可以在串行接口的多种条件出现时产生。该中断通过用户定义的回调函数来处理，回调函数用函数 **vAHI_SiRegisterCallback( )**注册。这个中断处理程序应识别真正的 SI 中断源并对其进行操作。有关回调函数和中断源的详细内容，请分别参考附录 A.1 和附录 B.2。在上面的写过程和读过程中，相关的 SI 主机中断源指示了字节传输完成或仲裁丢失；
- **轮询**：为了确定字节传输何时结束，应用可以定期调用 **bAHI_SiMasterPollTransferInProgress( )**，它指示了 SI 总线上是否正在进行传输。

一旦中断或轮询指示字节传输已完成，就必须执行更多检查来确定主机是否应停止数据传输并释放 SI 总线：

1）在写从机的情况下，应用应调用函数 **bAHI_SiMasterCheckRxNack( )**来指示在字节传输后从从机接收到 ACK 还是 NACK：

- ACK 指示从机可以接收更多数据，因此可启动更多字节传输；
- NACK 指示从机不能再接收任何数据，因此数据传输必须停止且 SI 总线释放。

2）假设 SI 总线还没有被释放，应用应调用函数 **bAHI_SiMasterPollArbitrationLost( )**来检查 SI 主机是否已丢失 SI 总线的仲裁。如果是这样，就必须停止数据传输并释放 SI 总线。

通过调用函数 **bAHI_SiMasterSetCmdReg( )**来发布停止命令，数据传输被停止且 SI 总线释放。

13.2 SI从机

JN516x 器件的 SI 外设可用作 SI 主机或 SI 从机（但不同时用作主机或从机）。本节描述了允许 SI 从机参与由远程 SI 主机启动的数据传输所必须执行的操作。

13.2.1 使能SI从机及其中断

SI 从机必须先使用函数 **vAHI_SiSlaveConfigure( )**进行配置和使能。这个函数要求 SI 从机的地址大小被指定为 7-位或 10-位，并指定 SI 从机地址本身。该函数还允许配置 SI 从机中断的发生，在下列条件出现时可触发中断：

- 数据缓冲区需要数据字节发送到 SI 主机；
- 数据缓冲区中的字节发送到 SI 主机，缓冲区有空位接收下一个字节；
- 数据缓冲区含有来自 SI 主机的数据字节，可以通过 SI 从机读取；
- 从 SI 主机接收到的最后的数据字节（数据传输的末尾）；
- I^2C 协议错误。

SI 中断（属于类型 E_AHI_DEVICE_SI）通过用户定义的回调函数来处理，回调函数使用函数 **vAHI_SiRegisterCallback( )**注册。这是与用于 SI 主机相同的注册函数。有关回调函数原型的详细内容，请参考附录 A.1。

警告:

已注册的回调函数只在 RAM 保持供电的睡眠模式期间保留。如果在睡眠过程中 RAM 停止供电并且需要中断，必须先注册回调函数，然后再在唤醒时调用 u32AHI_Init( )。

vAHI_SiSlaveConfigure( )还允许脉冲抑制滤波器使能，该滤波器抑制在时钟和数据线上脉宽少于 62.5ns 的任何假脉冲（高或低）。同时要注意的是，使用该函数使能的 SI 从机稍后可以使用 **vAHI_SiSlaveDisable( )**禁能。

使能时，接口将 DIO14 管脚用作时钟线，DIO15 管脚用作双向数据线（但不会给时钟供电）。这些信号可以使用函数 **vAHI_SiSetLocation( )** 分别传输到 DIO16 和 DIO17。

13.2.2 接收SI主机的数据

SI 主机指示它需要发送数据到特定的 SI 从机，如第 13.1.2 节所述。SI 从机自动根据这个请求的协议来响应 SI 主机，但与从机相关的应用必须处理来自主机的数据。

SI 总线上的数据传输由一系列数据字节组成，其中必须接收每个字节并从 SI 从机中读取，然后再接收下一个字节。中断用于告知来自 SI 主机的数据字节到达。

- 当来自 SI 主机的数据字节到达并可从 SI 从机缓冲区中读取时可以产生一个中断；
- 当来自 SI 主机的最后传输的数据字节到达并可从 SI 从机缓冲区中读取时可以产生一个中断。

为了使用这些中断，在调用函数 **vAHI_SiSlaveConfigure( )**时它们必须已使能。注册的 SI 中断处理程序也必须处理这些中断，见第 13.2.1 节。

一旦接收到的数据字节可以在 SI 从机的缓冲区中使用，它就可以从缓冲区中读取，该操作通过应用使用函数 **u8AHI_SiSlaveReadData8( )**来实现。

13.2.3 发送数据到SI主机

SI 主机指示它需要从特定的 SI 从机中获取数据，如第 13.1.3 节所述。SI 从机根据这个请

求的协议自动响应 SI 主机，但从机相关联的应用必须提供被发送到主机的数据。

SI 总线上的数据传输由一系列的数据字节组成，其中的每个字节必须被写入 SI 从机的缓冲区并且在下一个字节写入缓冲区之前发送。使用中断来通知缓冲区何时需要下一个数据字节。为了使用这些中断，在调用函数 **vAHI_SiSlaveConfigure( )**时必须使能这些中断。注册的 SI 中断处理程序必须处理这些中断，见第 13.2.1 节。

一旦在 SI 从机的缓冲区中需要新的数据字节，就可以通过应用使用函数 **vAHI_SiSlaveWriteData8( )**将该字节写入缓冲区。

第14章 串行外设接口（SPI）主机

本章阐述了使用集成外设 API 函数来控制 JN516x 微控制器的串行外设接口（SPI）主机。

JN516x 微控制器的串行外设接口允许在微控制器和外围设备之间进行高速同步数据传输，而不需要软件介入。当微控制器用作 SPI 总线的主机时，所有其它连接到总线的设备在主机 CPU 的控制下都被认为是从机设备。

JN516x 器件的 SPI 主机设备支持多达 3 个从机。

注:

在 JN516x 器件上，SPI 主机默认情况下是禁能的，并且与其它功能共用管脚－这点与 JN514x 器件不同，JN514x 器件的 SPI 主机使用特定的管脚，它在复位后使能以便于从外部 Flash 器件中启动。

14.1 SPI总线连接

SPI 主机使用管脚 DO0 来输出时钟（SPICLK），使用 DO1 输入数据（SPIMISO），使用 DIO18 输出数据（SPIMOSI），这些信号在 SPI 总线上共用。

可使用 3 条从机-选择输出线：SPISEL0、SPISEL1 和 SPISEL2。如果使能，它们分别出现在 DIO19、DIO0 和 DIO1。但输出线 SPISEL1 和 SPISEL2 可使用函数 **vAHI_SpiSelSetLocation( )** 传输到 DIO14 和 DIO15。

14.2 数据传输

数据传输是全双工，因此由两个通信的设备同时发送数据。要发送的数据在设备的 FIFO 缓冲区（移位寄存器）中存储。可使用 1 位至 32 位（包含在内）之间的任何数据传输宽度。数据传输顺序可配置为先 LSB（最低位）或先 MSB（最高位）。

由于数据传输是同步的，因此发送和接收设备使用 SPI 主机提供的相同时钟。SPI 设备使用外设时钟（有关系统时钟选项，请见第 3.1 节），该时钟可递减分频并允许位速率高达 16Mbps。

可使能一个中断，该中断在数据传输完成时产生。

14.3 SPI模式

SPI 使用的操作模式（0、1、2 或 3）决定了数据锁存的时钟边沿，这由两个布尔参数、时钟极性和相位来确定，如下表所示。

表 14.1 SPI 操作模式

SPI 模式	极性	相位	描述
0	0	0	数据在时钟的上升沿锁存
1	0	1	数据在时钟的下降沿锁存
2	1	0	时钟翻转且数据在时钟的下降沿锁存
3	1	1	时钟翻转且数据在时钟的上升沿锁存

14.4 从机选择

在传输数据之前，SPI 主机必须选择它想要通信的从机。因此，相应的从机选择线必须有效。通常 SPI 主机一次与一个从机进行通信，因此不会同时接收多个从机的数据（除非可阻止从机设备发送数据）。所提供的“自动从机选择特性”在数据传输期间仅使所选的从机选择线有效。

当特定的从机设备执行多个连续的数据传输时，手动进行从机选择比“自动从机选择”更

好，这使从机无需被取消选择，然后在相邻的传输之间重新选择。

14.5 使用串行外设接口

本节阐述了如何使用集成外设 API 函数来操作串行外设接口。

14.5.1 执行数据传输

如下执行 SPI 数据传输：

1）必须先使用函数 **vAHI_SpiConfigure( )**来配置和使能 SPI 主机。这个函数允许配置：

- SPI 从机数；
- 时钟分频器（用于外设时钟）；
- 数据传输顺序（先 LSB 还是先 MSB）；
- 时钟极性（不变还是翻转）；
- 相位（在时钟的上升沿锁存数据还是在下降沿锁存数据）；
- 自动从机选择；
- SPI 中断。

如果 SPI 中断使能，则必须使用函数 **vAHI_SpiRegisterCallback( )**来注册相应的回调函数，见 14.6 节。

2）必须使用函数 **vAHI_SpiSelect( )**来选择 SPI 从机。如果“自动从机选择”关闭，则相应的从机选择线将立即有效，否则这些线将仅在后续的数据传输期间有效。

3）使用 **vAHI_SpiStartTransfer( )**来执行数据传输。传输大小可指定为 1 位至 32 位。

4）通过等待 SPI 中断（如果使能）指示完成，或通过调用 **vAHI_SpiWaitBusy( )**在传输完成时返回，或通过周期性调用 **bAHI_SpiPollBusy( )**检查 SPI 主机是否仍然忙来完成传输。

5）使用 **u32AHI_SpiReadTransfer32( )**来读取从从机接收的数据。读取的数据与返回的 32-位值的右边（较低位）对齐。

6）如果要求另一个传输，则下一个数据必须重复步骤 3 至步骤 5。否则，如果“自动从机选择”关闭，则 SPI 从机必须通过调用 **vAHI_SpiSelect(0)或 vAHI_SpiStop( )**来取消选择。

许多其它的 SPI 函数存在于集成外设 API 中。使用函数 **vAHI_SpiReadConfiguration( )**可获取和保存当前的 SPI 配置。若有需要，使用函数 **vAHI_SpiRestoreConfiguration( )** 可稍后在 SPI 中恢复保存的配置。

14.5.2 执行一个连续传输

通过调用函数 **vAHI_SpiContinuous( )**而不是 **vAHI_SpiStartTransfer( )**来启动连续的 SPI 传输。这种模式有助于连续读取接收到的数据。输进来的数据传输由硬件自动控制，接收数据，硬件等待这个数据被软件读取，再允许执行下一个进来的数据传输。

在这种情况下，第 14.5.1 节中步骤 1-2 的过程保持相同，但步骤 3 以及之后的步骤由下面的步骤来替代：

3）使用 **vAHI_SpiContinuous( )**来启动一个连续的数据传输，这要求指定单次传输的数据长度（1 位至 32 位）。

4）必须周期性调用 **bAHI_SpiPollBusy( )**来检查 SPI 主机是否仍忙于处理单个传输。

5）一旦完成最近的传输（SPI 主机不再忙），那么从这个传输中接收到的数据必须通过调用函数 **u32AHI_SpiReadTransfer32( )**来读取，读取的数据与返回的 32-位值的右边（较低位）对齐。

6）一旦数据已读取，下一次传输将自动发生且传输的数据必须被读取，详细内容如上面的步骤 4-5 所述。但是，通过再次调用函数 **vAHI_SpiContinuous( )**可随时停止连续传输，这个时候连续模式被禁能（调用这个函数后，在传输停止之前将会再有一次传输）。

7）如果“自动从机选择”关闭，在停止一个连续传输后，必须通过调用 **vAHI_SpiSelect(0)** 来取消选择 SPI 从机。

14.6 SPI中断

SPI 中断可用于指示由 SPI 主机启动的数据传输何时完成。这个中断在 **vAHI_SpiConfigure()** 中使能。

SPI 中断由用户定义的回调函数来处理，回调函数用函数 **vAHI_SpiRegisterCallback( )**来注册。当出现 E_AHI_DEVICE_SPIM 类型的中断时，自动调用相应的回调函数。有关回调函数原型的详细内容，请参考附录 A.1。

警告:

已注册的回调函数只在 RAM 保持供电的睡眠模式期间保留。如果在睡眠过程中 RAM 停止供电并且需要中断，必须先注册回调函数，然后再在唤醒时调用 u32AHI_Init( )。

第15章 串行外设接口（SPI）从机

本章阐述了使用集成外设 API 函数来控制 JN516x 微控制器的串行外设接口（SPI）从机。

JN516x 微控制器的串行外设接口允许在微控制器和外围设备之间进行高速同步数据传输，而不需要软件介入。

注:

JN516x 微控制器的 SPI 主机设备在第 14 章中描述。

15.1 SPI从机操作

SPI 从机用来执行 JN516x 微控制器和“远程”处理器之间的高速数据交换，这个处理器是无线网络节点中包含的一个独立的处理器。远程处理器必须包含一个启动数据传输的 SPI 主机设备。数据交换并要求最少地使用 CPU。数据传输是全双工，因此两个通信的设备同时发送和接收数据。

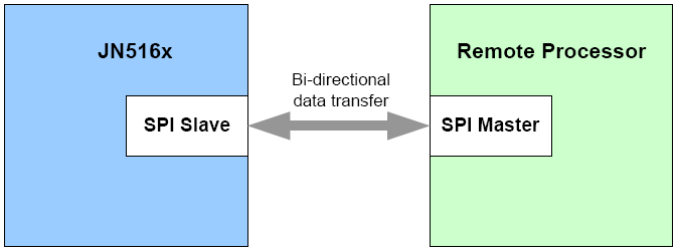

图 15.1 JN516x SPI 从机

SPI 从机使用系统 RAM 中独立可配置的 FIFO 缓冲区来存储要发送和接收的数据字节。

警告:

仅支持 SPI 模式 0。在数据链接的两端，要发送的数据在时钟下降沿上改变，要接收的数据在时钟上升沿采样。

15.1.1 SPI总线连接和DIO使用

SPI 从机使用下列总线连接：

- 从机时钟输入，SPISCLK；
- 从机数据输出，SPISMISO；
- 从机数据输入，SPISMOSI；
- 从机选择输入，SPISSEL。

这些信号使用下列 DIO 管脚：

- SPISCLK 使用 DIO15；
- SPISMOSI 和 SPISMISO 使用 DIO12-13 或者 DIO16-17；
- SPISSEL 使用 DIO14。

当调用 **bAHI_SpiSlaveEnable( )**时配置用于 SPISMOSI 和 SPISMISO 的 DIO 管脚。

15.1.2 SPI从机FIFO和中断

SPI 从机设备的数据输入（接收）和数据输出（发送）通道包含 FIFO 缓冲区，这些缓冲区位于 RAM 中。当 SPI 从机使用函数 **bAHI_SpiSlaveEnable( )**初始化时，这些缓冲区的确切位置和大小由应用来定义。每个缓冲区的大小可高达 255 字节。

必须指定填充-级别阈值（以字节为单位），这个阈值提示应用写数据到发送缓冲区以及从接收缓冲区中读取数据。

- 对于发送 FIFO，这个阈值被认为是足够低的填充-级别，可以使更多数据写入缓冲区。如果中断使能，则在缓冲区的数据量下降到这个级别以下时产生一个中断；
- 对于接收 FIFO，这个阈值被认为是足够高的填充-级别，可以从缓冲区中读取数据。如果中断使能，则在缓冲区的数据量上升到这个级别以上时产生一个中断。

在上面的函数调用中还必须指定接收超时时间（以微秒为单位）。在 SPI 传输结束后，如果接收 FIFO 在这个持续时间保持不为空，那么将产生一个超时中断（如果使能）来提示应用从缓冲区中读取数据。这防止接收的数据在缓冲区中保留太长时间而没有被读取。

SPI 从机中断必须使能以便于使用上述的缓冲区阈值和接收超时。此外，当使用函数 **bAHI_SpiSlaveEnable( )**配置设备时可使能中断。如果这些中断使能，那么处理 SPI 从机中断的用户定义回调函数必须使用函数 **vAHI_SpiSlaveRegisterCallback( )**来注册。当出现 E_AHI_DEVICE_SPIS 类型的中断时自动调用回调函数。有关回调函数原型的详细内容，请参考附录 A.1。

警告:

已注册的回调函数只在 RAM 保持供电的睡眠模式期间保留。如果在睡眠过程中 RAM 停止供电并且需要中断，必须先注册回调函数，然后再在唤醒时调用 u32AHI_Init( )。

15.2 使用SPI从机

如下所示，通过 SPI 从机执行数据传输（这个过程假设 SPI 从机中断使能）：

1）必须先使用函数 **bAHI_SpiSlaveEnable( )**初始化和配置 SPI 从机。这个函数允许以下配置：

- 发送/接收 SPI 数据的位顺序（LSB 先或 MSB 先）；
- 用于 SPISMISO 和 SPISMOSI 的 DIO 管脚；
- 发送 FIFO 缓冲区（包括 RAM 的起始地址），大小（以字节为单位）和写阈值（以字节为单位），见第 15.1.2 节；
- 接收 FIFO 缓冲区（包括 RAM 的起始地址），大小（以字节为单位），读阈值（以字节为单位）和超时（以微秒为单位），见第 15.1.2 节；
- SPI 从机中断（该中断应被使能）。

2）处理 SPI 从机中断的用户定义回调函数必须用函数 **vAHI_SpiSlaveRegisterCallback( )** 来注册。

3）应用现在可以将发送数据装入发送 FIFO（当远程 SPI 主机启动传输时将发送数据）：

a) 必须使用函数 **vAHI_SpiSlaveTxWriteByte( )**将原始数据写入发送 FIFO。写入的字节数必须不能超过缓冲区的大小。默认情况下，如果发送 FIFO 为空且传输由远程 SPI 主机启动，则 SPI 从机将发送数据字节 0x00。

b) 随后，应用必须等待一个写阈值中断来进一步提示写入发送 FIFO。当出现这个中断时，将调用用户定义的回调函数来处理中断，**vAHI_SpiSlaveTxWriteByte( )**在这个回调函数中被调用。写入的字节数不应超出缓冲区大小减去缓冲区写阈值的结果。

4）应用现在可以从接收 FIFO 中读取任何接收的数据。为了完成这个操作，它应等待一个读阈值中断或读超时中断。当出现其中一个中断时，将调用用户定义的回调函数来处理中断，函数 **u8AHI_SpiSlaveRxReadByte( )**在这个回调函数中被调用。

u8AHI_SpiSlaveTxFillLevel( )、**u8AHI_SpiSlaveRxFillLevel( )**和 **u8AHI_SpiSlaveStatus( )**

函数可以使能应用以非中断驱动的方式来监控 SPI 从机。

提示:

尽管数据传输是全双工，但通过在不想要的方向传输伪数据也可以实现单工传输。

第16章 Flash存储器

本章阐述了使用集成外设 API 函数来控制 Flash 存储器。

JN516x 微控制器具有片上 Flash 存储器。这种非易失性存储器用于存储二进制应用和相关的应用数据。JN516x 器件也可以选择连接到外部 Flash 存储器设备。

集成外设 API 包含了允许应用擦除、编程和读 Flash 存储器扇区的函数。通常，这些函数用于存储和取出应用数据，这可能包含了进入 RAM 不保持供电的睡眠模式之前在非易失性存储器中保存的数据。

16.1 Flash存储器构造和类型

Flash 存储器被划分为多个扇区。扇区数取决于 Flash 器件类型，但应用二进制通常存储在第一个扇区（也就是扇区 0）的起始位置，应用数据存储在最后的扇区中。空的（没有数据）Flash 存储器扇区内容全部是二进制 1。当数据被写入扇区时，相应的位从 1 变为 0。

下列表格给出了片上 Flash 存储器的细节，以及所支持的 JN516x 系列微控制器的外部 Flash 器件。

表 16.1 片上 Flash 存储器

JN516x 芯片	扇区数	扇区大小（Kbytes）	总大小（Kbytes）
JN5168	8	32	256
JN5164	5	32	160
JN5161	2	32	64

表 16.2 支持的外部 Flash 器件

制造商	Flash 器件	扇区数	扇区大小（Kbytes）	总大小（Kbytes）
Atmel	AT25F512	2	32	64
STMicroelectronics	M25P05A	2	32	64
Microchip	SST25VF010A	4	32	128
STMicroelectronics	M25P10A	4	32	128
STMicroelectronics	M25P20	4	64	256
Winbond	W25X20B	4	64	256
STMicroelectronics	M25P40	8	64	512

16.2 API函数

所提供的 Flash 存储器函数可用于与片上 Flash 器件和任何兼容的外部 Flash 器件（详见第 16.1 节）交互。这些函数可以访问 Flash 存储器的任何扇区，应用从第一个扇区（0）开始存储，应用数据通常存储在最后的扇区。若要了解所需扇区的细节，请参考 Flash 器件的数据手册。Flash 存储器函数在第 31 章中全面介绍。

16.3 Flash存储器的操作

本节描述了如何使用 Flash 存储器函数来擦除、读和写 Flash 存储器扇区。

调用的第一个 Flash 存储器函数必须是初始化函数 **bAHI_FlashInit( )**。在外部 Flash 存储器的情况下，这个函数要求指定附加的 Flash 器件类型。

也可以指定自定义的外部 Flash 器件。在这种情况下，必须提供一系列自定义函数，API

将使用这些函数来访问 Flash 器件。

注：可使用函数 **bAHI_FlashEECerrorInterruptSet( )**来使能中断，这些中断在片上 Flash 器件出现错误时产生。当出现 Flash 存储器中断时，调用已注册的用户定义回调函数。

16.3.1 擦除Flash存储器的数据

擦除 Flash 存储器的一部分就是将任意位 0 设为位 1。函数 **bAHI_FlashEraseSector( )**可用于擦除 Flash 存储器的整个扇区。任何扇区均可被擦除。

警告:

注意不要擦除基本数据，比如说应用代码。应用在片上 Flash 存储器的起始位置（从扇区 0 开始）存储。

注:

JN516x 器件的内部 Flash 存储器有大概 100ms 的扇区擦除时间。

16.3.2 读Flash存储器的数据

函数 **bAHI_FullFlashRead( )**可用于读取 Flash 存储器任何扇区的数据。这个函数可用来读取扇区任何位置的数据。

16.3.3 写数据到Flash存储器

在写第一个数据到 Flash 存储器的扇区之前，扇区必须为空（内容由全部二进制 1 组成），因为写操作只会将 1 变为 0（相应的地方）。因此，在写第一个数据到扇区前有必要擦除相关的扇区，如第 16.3.1 节所述。

函数 **bAHI_FullFlashProgram( )**可用于写数据到 Flash 存储器的任何扇区。这个函数可用来写扇区任意位置的数据。当添加数据到扇区时，必须确保扇区的相应部分已为空（内容包含所有二进制 1）。

有一个方法确保数据被成功添加到扇区，如下所示：

1）将整个扇区读入 RAM 中（见第 16.3.2 节）。

2）擦除 Flash 存储器的整个扇区（见第 16.3.1 节）。

3）添加新数据到 RAM 的现有数据中。

4）将这个数据写回 Flash 存储器的扇区中。

警告:

在 JN516x 器件的内部 Flash 存储器中，每个扇区被划分为 16-字节的页字（pagewords）。必须不能执行写入非空页字的操作，含有非空页字的扇区应该先用函数 **bAHI_FlashEraseSector( )**进行擦除，然后再写入页字。如果用户忽略了扇区擦除操作，那么在读取页字时将有可能导致后面的错误，这个读错误将触发一个中断，并执行使用函数 **bAHI_FlashEECerrorInterruptSet( )**注册的回调函数。

JN516x 器件内部 Flash 存储器的每个扇区可以写/擦除 10000 次。有关外部 Flash 存储器的擦除次数请参考器件特定的数据手册。

注:

JN516x 器件的内部 Flash 存储器有大概 1ms 的扇区写时间。

16.4 控制外部Flash存储器的功率

当 JN516x 微控制器在睡眠模式下（包括深度睡眠）时，可以关断任何外部 Flash 存储器。在睡眠期间 Flash 器件不供电可以节省更多功耗并延长电池寿命。

有两个函数（见下面的描述）可以控制外部 Flash 器件的功率，但它们只适用于下面的

STMicroelectronics 器件：

- STM25P05A
- STM25P10A
- STM25P20
- STM25P40

为 Flash 器件调用这些函数将没有作用。

下面列出了在进入睡眠模式前和进入睡眠模式后所需的函数调用。

在进入睡眠模式前

在进入睡眠模式前通过调用函数 **vAHI_FlashPowerDown( )**可以将上述外部 Flash 存储器器件的电源关断。在调用 **vAHI_Sleep( )**之前必须调用这个函数。

在进入睡眠模式后

如果在进入睡眠模式前使用函数 **vAHI_FlashPowerDown( )**来关断 Flash 存储器器件电源，那么从睡眠模式唤醒时必须调用函数 **vAHI_FlashPowerUp( )**使 Flash 存储器器件再次通电。

提示:

为了节省功耗，您可以在 JN516x 启动时关断外部 Flash 存储器器件的电源，并且仅在需要的时候使 Flash 器件上电。