睡眠及唤醒使用指南

概述

本手册适用范围：

适用范围
SPC1169，SPD1179，SPD1176

本手册以 **SPD1179/SPD1176, SPC1169** 为例：

- 文中有关 Active, Stop, Sleep 的描述以及状态切换因 **SPD1179/SPD1176, SPC1169** 不同而具有本质差别，详见正文。

SPINTROL

目录

SPINTROL

©2023 旋智电子科技（上海）有限公司

图片列表

SPINTROL

表格列表

SPINTROL

©2023 旋智电子科技（上海）有限公司

版本历史

版本	日期	作者	状态	变更
A/0	2023 年 11 月 30 日	Hang Su	Released	首次发布。

SPINTROL

术语或缩写

术语或缩写	描述
LIN	Local Interconnect network，低成本串行通信网络
MON	指 SPD1179 中的 MON 管脚

SPINTROL

©2023 旋智电子科技（上海）有限公司

1 SPD1179/SPD1176 电源管理模块（PMU）

电源管理模块为 MCU 部分（DVDD5，DVDD33，VCAP12）和 VDD5EXT 提供所需要的供电。电源管理系统确保了 SPD1179/SPD1176 的行为安全，并具有一个状态机来控制电源模式转换。SPD1179/SPD1176 实现了具有以下特性的电源管理模块：

支持三种电源模式（active、stop 和 sleep）

Active（正常工作模式）

DVDD5，DVDD33 和 VCAP12 由片内 LDO 以满载能力供电，所有时钟根据用户配置开或关。

Stop（停止模式）

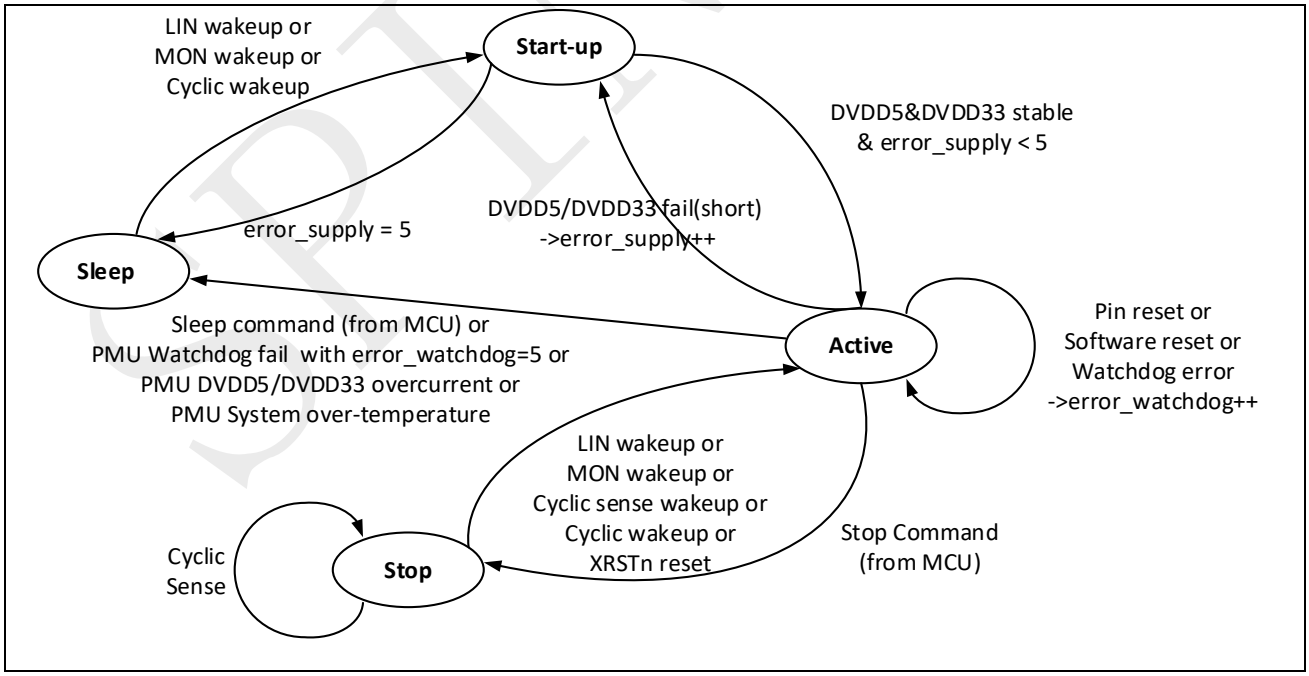
除了用于电源模式状态机的 100KHz 时钟外，其余所有时钟关闭。DVDD5, DVDD33 和 VCAP12 由低功耗低载能力的 LDO 供电，VCAP12 可以根据用户配置降至 1.2V 以下来进一步降低静态电流。所有寄存器和存储器保持内容，系统可以被不同的唤醒源唤醒，比如，XRSTn、LIN、MON，cyclic-wakeup 和 cyclic-sense-wakeup（可以配置为任意 GPIO 引脚上的任意电平）。CPU 将直接继续执行停止时的下一条指令（XRSTn 唤醒将引起系统复位）。

Sleep（睡眠模式）

DVDD5，DVDD33 和 VCAP12 完全断电，系统可以通过 LIN/MON/cyclic-wakeup 唤醒，并开始冷启动过程。

三种电源模式转换如图 1-1 所示。

图 1-1：电源模式转换



1.1 睡眠模式

从图 1-1 可以看出，导致进入睡眠模式的事件有：睡眠命令、**PMU WDT 错误、PMU DVDD5/DVDD33 过流、PMU 系统过温**，其中 **PMU WDT 错误、PMU DVDD5/DVDD33 过流、PMU 系统过温发生时，硬件将会自动迫使电源管理模块进入睡眠模式**，不需要软件做任何设置，本章以下几个小结将对这其余两种诱因的使用方式进行详细的描述。

需要特别强调的是，当唤醒条件有效时，即使发生如上描述的四种睡眠事件，电源管理模块也不会进入睡眠模式。

1.1.1 睡眠命令

此种模式是通过软件发送睡眠指令，主动迫使电源管理模块进入睡眠模式，如下代码演示了如何使用代码发送睡眠命令主动进入睡眠模式。

Example Code

```
#include "spd1179.h"
#include <stdio.h>

ErrorStatus              eErrorState;

int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    /* Enable sleep/stop command */
    eErrorState = EPWR_WriteRegister(HV_REG_CTLKEY, KEY_PMU_CMD);
    if (eErrorState == ERROR)
    {
        printf("Write CTLKEY register FAIL\n");
        return 0;
    }

    pHWLIB->SYSTEM_Sleep();

    while (1)
    {

    }
}
```


©2023 旋智电子科技（上海）有限公司

1.1.2 系统过温

SPD1179 内置有一个温度传感器，当芯片温度超过某一温度（记为 OT_fatal）时，电源管理模块将会进入睡眠模式，这一动作由硬件自动完成，软件唯一需要做的是设置一个发出该动作的温度阈值。温度阈值 OT_fatal 有 7 个档位可以选择（详细请查阅 Technical Reference Manual 中 EVTTHCTL0 寄存器），且通过如下所示的代码进行设置。

Example Code

```
#include "spd1179.h"
#include <stdio.h>

ErrorStatus          eErrorState;           /* Function State */

int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("Enter the test\n");

    /* HV init */
    eErrorState = HV_Init(&u16PREDRIID);
    if (eErrorState == ERROR)
    {
        printf("Init HV mode FAIL\n");
        return 0;
    }else{
        printf("Init HV mode SUCCESS[ID:%d]\n", u16PREDRIID);
    }

    /* HV parameter write enable */
    eErrorState = EPWR_WriteRegister(HV_REG_CTLKEY, KEY_USER_REG);
    if (eErrorState == ERROR)
    {
        printf("Write CTLKEY register FAIL\n");
        return 0;
    }

    /* Set the PMU over-temperature threshold */
    eErrorState = EPWR_WriteRegisterField(HV_REG_EVTTHCTL0,
EVTTHCTL0_PMUOTWARN_Msk, EVTTHCTL0_PMUOTWARN_129C);
    if (eErrorState == ERROR)
    {
        printf("EPWR_WriteRegisterField FAIL\n");
        return 0;
    }
    while (1)
    {
    }
}
```

1.2 停止模式

如第一章节开始部分所述，进入停止模式之后，DVDD5, DVDD33 和 VCAP12 由低功耗低载能力的 LDO 供电，可以看出，此时仍然维持着对芯片的供电，以为着 RAM 的数据不会丢失。但需要注意的是，此时 MCU 没有任何时钟，外设的一切功能将停止（例如：PWM 将不能产生波形等），且此时的 MCU 也不能捕捉 GPIO 等外设的瞬时信号（例如，在停止模式时，某个 GPIO 有上升沿产生，此时 MCU 并不知道有这个事件发生，也就不能在唤醒之后进入对应的 GPIO 边沿中断）。

从图 1-1 中可以看出，通过软件方式向高压部分发送停止指令是进入停止模式的唯一途径，如下代码演示了如何使用代码发送停止命令进入停止模式。

Example Code

```
#include "spd1179.h"
#include <stdio.h>

ErrorStatus              eErrorState;

int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    /* Enable sleep/stop command */
    eErrorState = EPWR_WriteRegister(HV_REG_CTLKEY, KEY_PMU_CMD);
    if (eErrorState == ERROR)
    {
        printf("Write CTLKEY register FAIL\n");
        return 0;
    }

    pHWLIB->SYSTEM_Stop();

    while (1)
    {

    }
}
```


©2023 旋智电子科技（上海）有限公司

1.3 唤醒

如概述部分图 1-1 的描述，在睡眠模式和停止模式下，两者可供选择的唤醒源不同。

睡眠模式：可以通过 LIN/MON/Cyclic-WakeUp 唤醒，并开始冷启动过程，冷启动意味着将重新开始执行代码。

停止模式：电源管理模块可以被不同的唤醒源唤醒，比如，XRSTn、LIN、MON、cyclic-wakeup 和 cyclic-sense-wakeup（可以配置为任意 GPIO 引脚上的任意电平），其中，用 LIN、MON、cyclic-wakeup 和 cyclic-sense-wakeup 方式唤醒之后，将继续执行停止时的下一条指令，而用 XRSTn 唤醒之后，Core 将重启，这将导致代码重新开始执行。

1.3.1 睡眠模式唤醒

本小结将给如何使用如上文描述的三种唤醒睡眠模式的代码，代码中已使用宏区分开属于三种不同的唤醒方式的代码，用户可根据自己的需要进行选择。

- **LIN 唤醒**

若用户选择使用 LIN 作为唤醒源，将 LIN 管脚接地，并维持低电平大于 80us，将会唤醒电源管理模块，在验证示例代码时，若测试成功，将会看到“Init PRE-DRIVER mode SUCCESS”的串口打印字样。

注意：	1. LIN 协议要求从机要能识别>150us 的唤醒信号，按照 80us 规格设计的 SPD1179/SPD1176 完全满足 LIN 协议的要求。

- **MON 唤醒**

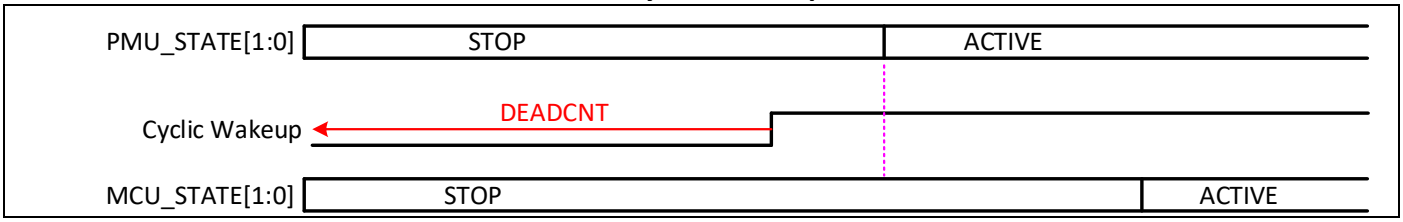
若用户选择使用 MON 作为唤醒源，由于在示例代码中已经将 MON 唤醒极性设置为高，所以在验证示例代码时，需要将 MON 管脚接到高电平，并维持超过 40us，将会唤醒电源管理模块，若测试成功，将会看到“Init PRE-DRIVER mode SUCCESS”的串口打印字样。

- **Cyclic-WakeUp 唤醒**

若用户使用 Cyclic-WakeUp 方式唤醒，在示例代码中已设置 DEADCNT 为 12ms，这意味着，电源管理模块将以 12ms 为周期，循环发出唤醒信号，使自己从停止模式转入 Active 模式。在验证示例代码时，若测试成功，将会看到“Init PRE-DRIVER mode SUCCESS”的串口打印字样。

Cyclic-WakeUp 图示如图 1-3：Cyclic-WakeUp 图示图 1-2 所示。

图 1-2：Cyclic-WakeUp 图示



注意：	1. 睡眠模式唤醒后的启动流程，和冷启动一致，所以测试时，需要注意，保持 BOOT 脚为低电平，此时才会执行正常的额启动流程，否则，芯片将一直卡死在 ROM 代码段。

Example Code

```
#include "spd1179.h"
#include <stdio.h>


uint32_t                u32PWMPeriod;           /* PWM Period*/
uint16_t                u16PREDRIID;           /* PRE-DRIVER mode ID */
ErrorStatus             eErrorState;
#define                   Wake_Up_Mode                          3


int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("Enter the test\n");

    /* HV init */
    eErrorState = HV_Init(&u16PREDRIID);
    if (eErrorState == ERROR)
    {
        printf("Init PRE-DRIVER mode FAIL\n");
        return 0;
    }
    else
    {
        printf("Init PRE-DRIVER mode SUCCESS[ID:%d]\n", u16PREDRIID);
    }

    /* HV parameter write enable */
    eErrorState = EPWR_WriteRegister(HV_REG_CTLKEY, KEY_USER_REG);
    if (eErrorState == ERROR)
    {
        printf("Write CTLKEY register FAIL\n");
        return 0;
    }

    #if(Wake_Up_Mode == 0) //LIN wake up setting
    eErrorState = EPWR_WriteRegisterField(HV_REG_PMUCTL, PMUCTL_LINWKUPEN_Msk | PMUCTL_CYCWKUPEN_Msk,
PMUCTL_LINWKUPEN_ENABLE | PMUCTL_CYCWKUPEN_DISABLE);
    if (eErrorState == ERROR)
    {
        printf("Write PMUCTL register FAIL\n");
        return 0;
    }
    #elif(Wake_Up_Mode == 1) //MON asynchronous wake up setting
    eErrorState = EPWR_WriteRegisterField(HV_REG_PMUCTL, PMUCTL_MONWKUPEN_Msk,
PMUCTL_MONWKUPEN_ENABLE);
    if (eErrorState == ERROR)
```


©2023 旋智电子科技（上海）有限公司

```
    {
        printf("Write PMUCTL register FAIL\n");
        return 0;
    }
    eErrorState = EPWR_WriteRegisterField(HV_REG_MONCTL, MONCTL_WKUPPOL_Msk | MONCTL_EN_Msk,
MONCTL_WKUPPOL_ACTIVE_HIGH | MONCTL_EN_ENABLE);
    if (eErrorState == ERROR)
    {
        printf("Write MONCTL register FAIL\n");
        return 0;
    }

    #elif(Wake_Up_Mode == 2) //Cyclic wake up setting
    eErrorState = EPWR_WriteRegisterField(HV_REG_PMUCTL, PMUCTL_CYCWKUPEN_Msk,
PMUCTL_CYCWKUPEN_ENABLE);
    if (eErrorState == ERROR)
    {
        printf("Write PMUCTL register FAIL\n");
        return 0;
    }

    /* Dead time is (DEADCNT+1) * 2ms */
    eErrorState = EPWR_WriteRegisterField(HV_REG_CYCWKUPCTL, CYCWKUPCTL_DEADCNT_Msk,
CYCWKUPCTL_DEADCNT_(5));
    if (eErrorState == ERROR)
    {
        printf("Write CYCWKUPCTL register FAIL\n");
        return 0;
    }
    #endif

    /* Enable sleep/stop command */
    eErrorState = EPWR_WriteRegister(HV_REG_CTLKEY, KEY_PMU_CMD);
    if (eErrorState == ERROR)
    {
        printf("Write CTLKEY register FAIL\n");
        return 0;
    }

    pHWLIB->SYSTEM_Sleep();

    while (1)
    {

    }
}
```

1.3.2 停止模式唤醒

如本章节开始所述，停止模式下可选的唤醒源有五种，其中 XRSTn 属于硬件直接唤醒，不需要软件进行设置，所以将不会对此方式进行过多的说明。其它四种方式均在以下示例代码中已使用宏区分开属于四种不同的唤醒方式的代码，用户可根据自己的需要进行选择。

- **LIN 唤醒**

若用户选择使用 LIN 作为唤醒源，将 LIN 管脚接地，并维持低电平大于 80us，将会唤醒电源管理模块，在验证示例代码时，若测试成功，将会看到“total test passed”的串口打印字样。

注意：	1. LIN 协议要求从机要能识别>150us 的唤醒信号，按照 80us 规格设计的 SPD1179/SPD1176 完全满足 LIN 协议要求； 2. 默认状态下 CYCWKUPEN 和 LINWKUPEN 作为唤醒源处于使能状态，在测试 LIN 唤醒功能时，请关闭 CYCWKUPEN 功能，避免出现不必要的现象。

- **MON 唤醒**

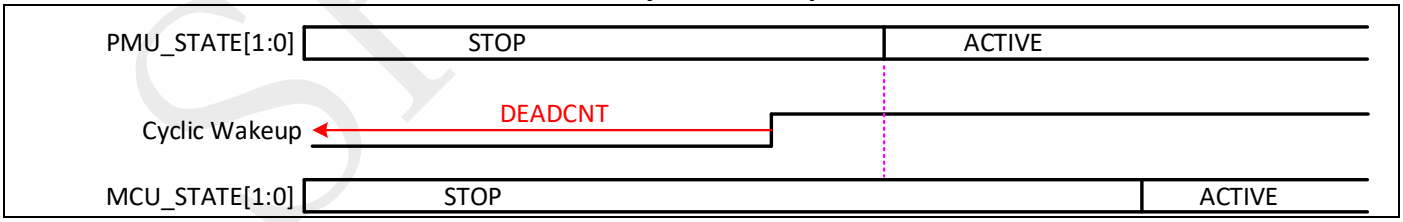
若用户选择使用 MON 作为唤醒源，由于在示例代码中已经将 MON 唤醒极性设置为高，所以在验证示例代码时，需要将 MON 管脚接到高电平，并维持超过 40us，将会唤醒电源管理模块，若测试成功，将会看到“total test passed”的串口打印字样。

- **Cyclic-WakeUp 唤醒**

若用户使用 Cyclic-WakeUp 方式唤醒，在示例代码中已设置 DEADCNT 为 12ms，这意味着，电源管理模块将以 12ms 为周期，循环发出唤醒信号，使自己从停止模式转入 Active 模式。在验证示例代码时，若测试成功，将会看到“total test passed”的串口打印字样。

Cyclic-WakeUp 图示如图 1-3 所示。

图 1-3：Cyclic-WakeUp 图示



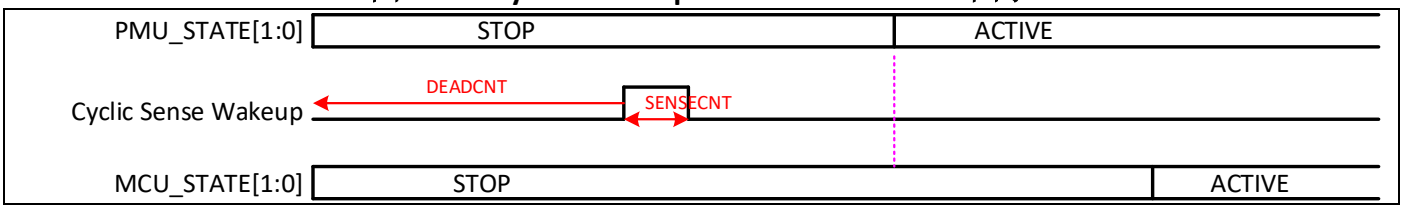
- **Cyclic-Sense-WakeUp 唤醒**

Cyclic-Sense-WakeUp 唤醒在实际的使用场景中，有三个参数可以设置，分别是：DEADCNT，SENSECNT，RAMPCNT。以下用两个场景对三个参数的功能加以说明。

场景一：电源管理模块以 DEADCNT 为周期，周期性开启以 SENSECNT 指定窗口时间的唤醒侦测窗口，在此窗口期内，检查在指定的 GPIO 上是否存在唤醒电平，若存在唤醒电平，则进入 Active 模式（若唤醒电平不在侦测窗口期内，则无法唤醒电源管理模块。）；若不存在，则继续保持停止模式。此时 Cyclic-Sense-WakeUp 的图示如图 1-4 所示。

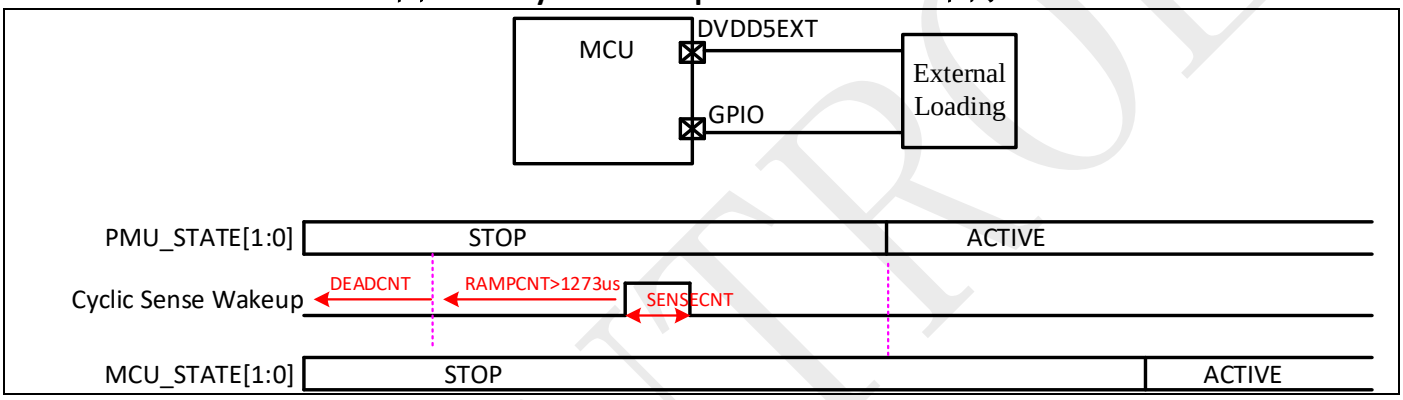
©2023 旋智电子科技（上海）有限公司

图 1-4：Cyclic-WakeUp Without DVDD5EXT 图示



场景二：外部负载需要通过本产品的 SVDD5EXT 供电，与此同时，GPIO 唤醒电平需要通过外部负载给出，在此应用场景中 Cyclic-Sense-WakeUp 的图示如图 1-5 所示。此种场景中，与场景一不同的是：在开启侦测窗口之前，需要设定 DVDD5EXT 的启动时间，此时间由 RAMPCNT 指定。需要注意的是，如图 1-5 所示，RAMPCNT 的时间最小需要设置为 1273us。

图 1-5：Cyclic-Wakeup With DVDD5EXT 图示



Example Code

```
#include "spd1179.h"
#include <stdio.h>


uint32_t            u32PWMPeriod;       /* PWM Period*/
uint16_t            u16PREDRIID;        /* PRE-DRIVER mode ID */
ErrorStatus         eErrorState;
#define             Wake_Up_Mode        0


int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("Enter the test\n");

    /* HV init */
    eErrorState = HV_Init(&u16PREDRIID);
    if (eErrorState == ERROR)
    {
        printf("Init PRE-DRIVER mode FAIL\n");
```

```
        return 0;
    }
    else
    {
        printf("Init PRE-DRIVER mode SUCCESS[ID:%d]\n", u16PREDRIID);
    }

    /* HV parameter write enable */
    eErrorState = EPWR_WriteRegister(HV_REG_CTLKEY, KEY_USER_REG);
    if (eErrorState == ERROR)
    {
        printf("Write CTLKEY register FAIL\n");
        return 0;
    }

    /* LIN wake up setting */
    #if(Wake_Up_Mode == 0)
    eErrorState = EPWR_WriteRegisterField(HV_REG_PMUCTL, PMUCTL_LINWKUPEN_Msk |
      PMUCTL_CYCWKUPEN_Msk, PMUCTL_LINWKUPEN_ENABLE | PMUCTL_CYCWKUPEN_DISABLE);
    if (eErrorState == ERROR)
    {
        printf("Write PMUCTL register FAIL\n");
        return 0;
    }

    /* MON asynchronous wake up setting */
    #elif(Wake_Up_Mode == 1)
    eErrorState = EPWR_WriteRegisterField(HV_REG_PMUCTL, PMUCTL_MONWKUPEN_Msk |
                                PMUCTL_LINWKUPEN_Msk | PMUCTL_CYCWKUPEN_Msk,
                                PMUCTL_MONWKUPEN_ENABLE | PMUCTL_LINWKUPEN_DISABLE |
                                PMUCTL_CYCWKUPEN_DISABLE);
    if (eErrorState == ERROR)
    {
        printf("Write PMUCTL register FAIL\n");
        return 0;
    }

    /* Wakeup level is high, MON pin need pull down */
    eErrorState = EPWR_WriteRegisterField(HV_REG_MONCTL, MONCTL_WKUPPOL_Msk |
                            MONCTL_EN_Msk | MONCTL_PULLMODE_Msk,
                            MONCTL_WKUPPOL_ACTIVE_HIGH | MONCTL_EN_ENABLE |
                            MONCTL_PULLMODE_PULL_DOWN);
    if (eErrorState == ERROR)
    {
        printf("Write MONCTL register FAIL\n");
        return 0;
    }

    /* Cyclic wake up setting */
    #elif(Wake_Up_Mode == 2)
    eErrorState = EPWR_WriteRegisterField(HV_REG_PMUCTL, PMUCTL_CYCWKUPEN_Msk |
                                    PMUCTL_LINWKUPEN_Msk, PMUCTL_CYCWKUPEN_ENABLE |
                                    PMUCTL_LINWKUPEN_DISABLE);
    if (eErrorState == ERROR)
    {
        printf("Write PMUCTL register FAIL\n");
        return 0;
    }

    /* Dead time is (DEADCNT+1) * 2ms */
    eErrorState = EPWR_WriteRegisterField(HV_REG_CYCWKUPCTL, CYCWKUPCTL_DEADCNT_Msk,
CYCWKUPCTL_DEADCNT_(5));
    if (eErrorState == ERROR)
    {
```

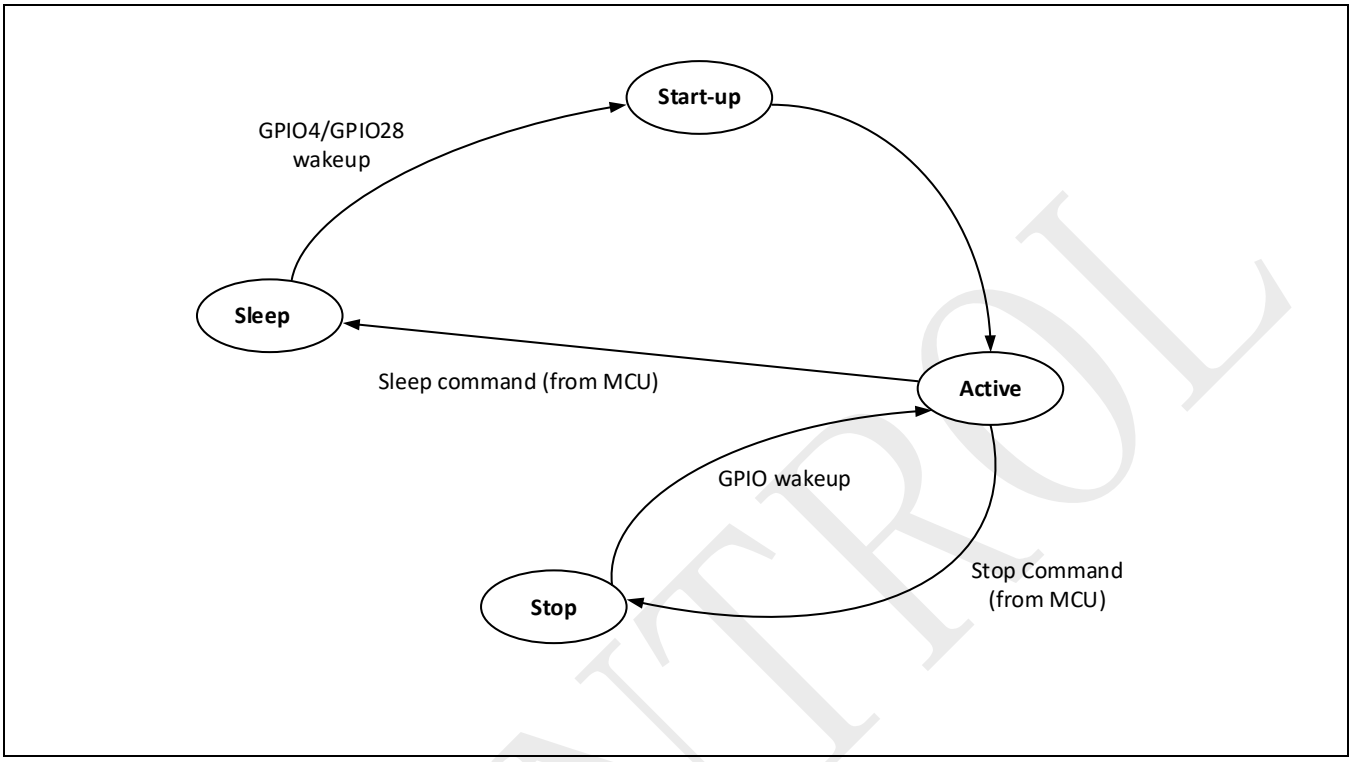

©2023 旋智电子科技（上海）有限公司

```
        printf("Write CYCWKUPCTL register FAIL\n");
        return 0;
    }

    /* Cyclic sense wake up setting */
    #elif(Wake_Up_Mode == 3)
    /* Use GPIO9 as the wake up source */
    SYSTEM_SetStopWakeUpByGPIO(PIN_GPIO9, LOW);

    eErrorState = EPWR_WriteRegisterField(HV_REG_PMUCTL, PMUCTL_CYCSENWKUPEN_Msk |
                    PMUCTL_LINWKUPEN_Msk | PMUCTL_CYCWKUPEN_Msk,
                    PMUCTL_CYCSENWKUPEN_ENABLE | PMUCTL_LINWKUPEN_DISABLE |
                    PMUCTL_CYCWKUPEN_DISABLE);
    if (eErrorState == ERROR)
    {
        printf("Write PMUCTL register FAIL\n");
        return 0;
    }
    /* Dead time is (DEADCNT+1) * 2ms, sense window is (SENSECNT+1) * 10us */
    eErrorState = EPWR_WriteRegisterField(HV_REG_CYCSENSECTL, CYCSENSECTL_DEADCNT_Msk
| \
            CYCSENSECTL_SENSECNT_Msk, CYCSENSECTL_DEADCNT_(10) |
CYCSENSECTL_SENSECNT_(15));
    if (eErrorState == ERROR)
    {
        printf("Write CYCSENSECTL register FAIL\n");
        return 0;
    }
    #endif

    /* Enable sleep/stop command */
    eErrorState = EPWR_WriteRegister(HV_REG_CTLKEY, KEY_PMU_CMD);
    if (eErrorState == ERROR)
    {
        printf("Write CTLKEY register FAIL\n");
        return 0;
    }

    pHWLIB->SYSTEM_Stop();

    printf("total test passed\n");

    while (1)
    {

    }
}
```

2 SPC1169 电源管理模块（PMU）

电源管理单元（PMU）负责为嵌入式 MCU（VCAP12）所需的所有电压供应。PMU 具有明确定义的状态机，用于控制电源模式的转换，如图 2-1 所示。

图 2-1：电源模式转换



SPC1169 提供的启动过程和 3 种电源模式的定义为：

Active 模式：

稳压器（VCAP12）由芯片内部具有全电流驱动能力的 LDO 供电；所有时钟根据用户在寄存器控制位中的设置而启用。

Stop 模式：

除了用于电源模式状态机的 128 KHz 时钟外，所有时钟都停止，VCAP12 由具有有限电流驱动能力的低功耗 LDO 供电，VCAP12 可以配置为低于 1.2V 以进一步降低静态电流，所有寄存器和内存保持内容不变。系统可以通过不同的 GPIO 进行唤醒，当退出停止模式时，CPU 将继续执行下一条指令。

Sleep 模式：

稳压器（VCAP12）完全关闭，它可以通过 GPIO4/GPIO28 被唤醒，整个系统将进行冷启动。

注意：	1. 在发送停止命令之前，用户应安全地禁用未使用的模拟模块； 2. 在停止模式下，泄漏电流在高温下显著增加，并且可能超过低功耗 LDO 的负载能力。在发出停止命令之前，用户需要确保结温低于 100°C。

©2023 旋智电子科技（上海）有限公司

2.1 睡眠模式

从图 2-1 可以看出，导致进入睡眠模式的事件只有睡眠命令。

需要特别强调的是，当唤醒条件有效时（见章节 2.3），即使运行睡眠指令，电源管理模块也不会进入睡眠模式。

如下代码演示了使用代码发送睡眠命令进入睡眠模式。

Example Code

```
pHWLIB->MCU_Sleep();
```

SPINTROL

2.2 停止模式

如文档概述部分所述，进入停止模式之后，VCAP12 由低功耗低载能力的 LDO 供电，可以看出，此时仍然维持着对芯片的供电，以为着 RAM 的数据不会丢失。但需要注意的是，此时 MCU 没有任何时钟，外设的一切功能将停止（例如：PWM 将不能产生波形等），且此时的 MCU 也不能捕捉 GPIO 等外设的瞬时信号（例如，在停止模式时，某个 GPIO 有上升沿产生，此时 MCU 并不知道有这个事件发生，也就不能在唤醒之后进入对应的 GPIO 边沿中断）。

从图 2-1 中可以看出，通过软件方式向高压部分发送停止指令是进入停止模式的唯一途径，如下代码演示了使用代码发送停止命令进入停止模式。

Example Code

```
pHWLIB->MCU_Stop();
```

SPINTROL

©2023 旋智电子科技（上海）有限公司

2.3 唤醒

如概述部分图 2-1 的描述，在睡眠模式和停止模式下，两者可供选择的唤醒源不同。

睡眠模式：可以通过 GPIO4/GPIO28 唤醒，并开始冷启动过程，冷启动意味着将重新开始执行代码。

停止模式：电源管理模块可以被不同的 GPIO 唤醒，唤醒之后，将继续执行停止时的下一条指令。

2.3.1 睡眠模式唤醒

示例演示通过 GPIO4 低电平唤醒，并开始冷启动过程（重新开始执行代码）。

注意：	1. 睡眠模式唤醒后的启动流程，和冷启动一致，所以测试时，需要注意，保持 BOOT 脚为低电平，此时才会执行正常的冷启动流程，否则，芯片将一直卡死在 ROM 代码段。

Example Code

```
#include <stdio.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif


ErrorStatus                eErrorState;



int main(void)
{
    CLOCK_InitWithRCO(100000000);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("Enter the test\n");

    /* Set the wakeup pin and level */
    eErrorState = SYSTEM_SetSleepWakeUpByGPIO(PIN_GPIO4, LOW);
    if (eErrorState == SUCCESS)
    {
        printf("SetSleepWakeUp passed\n");
    }
    else
    {
        printf("SetSleepWakeUp failed\n");
    }

    /* MCU sleep */
    pHWLIB->MCU_Sleep();

    while (1)
```

```
    {

    }
}
```

SPINTROL

©2023 旋智电子科技（上海）有限公司

2.3.2 停止模式唤醒

示例演示通过 GPIO0 低电平唤醒，唤醒之后，将继续执行停止时的下一条指令。

Example Code

```
#include <stdio.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif


ErrorStatus                 error_flag;


/*****************************************************************************
 *
 * @brief     In this case, we use the PIN_GPIO0 to wakeup the MCU from STOP state.
 *
 *****************************************************************************/


int main(void)
{
    CLOCK_InitWithRCO(100000000);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("Enter the test\n");

    /* Enable the PIN_GPIO0 LOW to wakeup the MCU */
    error_flag = SYSTEM_SetStopWakeUpByGPIO(PIN_GPIO0, LOW);
    if (error_flag == SUCCESS)
    {
        printf("SetStopWakeUp passed\n");
    }
    else
    {
        printf("SetStopWakeUp failed\n");
    }

    /* Wait until uart printf finished */
    Delay_Ms(10);

    while (1)
    {
        pHWLIB->MCU_Stop();

        /* the printf will be execute if we connect the PIN_GPIO0 low
        to wakeup the MCU from STOP state */
        printf("total test passed\n");

        /* Wait until uart printf finished */
        Delay_Ms(10);
```

```
    }
}
```

SPINTROL

©2023 旋智电子科技（上海）有限公司