SPD1078 低功耗模式使用指南

概述

在实际应用中，在处理器处于空闲模式时，常常需要让其进入低功耗模式，从而降低系统整体的功耗。通常来讲，处理器的功耗与时钟频率、工作电压以及外设是否工作有关。时钟频率越快，工作电压越高，工作的外设模块越多，处理器的功耗就越高；反之，则越低。

SPD1078 没有专门的低功耗模式设计，但是可以通过降低 CPU 和外设模块的时钟频率、关闭暂时不使用的外设模块和时钟信号、关闭 Gate Driver 的输出并进入 Standby 模式等方式达到降低芯片整体功耗的目的，实现芯片的低功耗模式。本文将介绍具体的实现过程和示例代码。

注意： 本文档主要以 SPD1078 为例进行介绍。

目录

图片列表

版本历史

版本	日期	作者	状态	变更
C/0	2024-02-27	周佳莉	Released	首次发布。

术语或缩写

术语或缩写	描述
CPU	Central Processing Unit，中央处理器

1 低功耗实现

1.1 实现方法

SPD1078 实现低功耗的模式的方法如下：

- 降低 CPU 和内部总线的时钟频率
- 降低处于工作状态的外设时钟频率
- 关闭暂时不用的外设模块
- 关闭暂时不用的时钟信号
- 关闭 Gate Driver 的输出，进入 Standby 模式

SPD1078 芯片的功耗主要由 CPU 和总线、外设模块、时钟信号以及 Gate Driver 的功耗构成。对这四者，要分别采取不同的方法降低其功耗：

- CPU 和总线：芯片内部总线时钟主要有 HCLK 和 PCLK，CPU 的时钟信号来自 HCLK，因此降低 HCLK 和 PCLK 的时钟频率就可以降低 CPU 和总线的功耗；
- 外设模块：对于需要工作的外设，降低其时钟频率；暂时不用的外设模块，关闭其时钟；
- 时钟信号：SPD1078 的时钟信号有 RCO0、RCO1、PLL 和 XO。若采用内部时钟，可以关闭 XO，其他时钟信号可以根据应用需要，选择性的关闭；
- Gate Driver：复位 Gate Driver，进入 Standby 模式，此时 Gate Driver 的输出处于关闭状态。

1.2 示例代码

本节通过一个具体的实例，来演示 SPD1078 如何进入或者退出低功耗模式。SPD1078 工作电压为 16V（PVDD0 和 PVDD1 均为 16V），通过 UART 接口接收命令，当接收到 SLEEP 命令（0x53），芯片进入低功耗模式；当接收到 WAKE 命令（0x57），芯片退出低功耗模式。

芯片接收到 SLEEP 命令后，采取的降低功耗的措施如下：

- 复位 Gate Driver，Gate Driver 的输出处于关闭状态；
- 将系统时钟切换到内部 RCO0 和 RCO1 时钟，RCO0 和 RCO1 的时钟频率都是 24MHz；
- 关闭 PLL 时钟和 XO 时钟；
- 关闭暂时不用的外设模块（SSP、QSPI、I2C、ADC、ECAP、PWM0~6、WDT0~1）；
- 降低 CPU 和总线的时钟频率，通过将系统时钟 63 分频实现，因此 CPU 和总线的时钟频率 = 24MHz / 63 = 381kHz；
- 降低 UART 模块的时钟频率，通过将系统时钟 63 分频实现，因此 UART 模块的时钟频率 = 24MHz / 63 = 381kHz。

具体代码如下：

Example Code

```
int main()
{
  uint8_t  u8Cmd = 0;
  uint8_t u8NewtonID;

  /* System Init */
  Sys_Init();

  /* Clock Init */
  CLOCK_InitWithRCO(CLOCK_HCLK_24MHZ);

  /* Delay Init */
  Delay_Init();


  /* UART Init */
  GPIO_SetPinChannel(GPIO_34, GPIO34_UART_TXD);
  GPIO_SetPinChannel(GPIO_35, GPIO35_UART_RXD);

  UART_Init(UART, 1200);

  printf("SPD1078 Low Power Mode Sample\n");


  /* Gate Driver Init */
  if(GATEDRV_Init(1000000, &u8NewtonID, 1000) == ERROR)
  {
    printf("GATEDRV Init Failed!\n");
    while(1);
  }

  /* Reset GATEDRV */
  if(GATEDRV_ResetREG() == ERROR)
  {
    printf("Reset GATEDRV Failed!\n");
    while(1);
  }

  /* Set GATEDRV output enable */
  GATEDRV_EnablePreDriverOutput();
  GATEDRV_EnableGate();


  while(1)
  {
    /* Wait for Command from UART */
    while(UART_IsRxDataReady(UART) == 0);

    /* Read Command from UART */
    u8Cmd = UART_ReadByte(UART);
```

```
    switch(u8Cmd)
    {
      case 'S':  /* SLEEP */

        /* Reset GATEDRV - Enter Standby mode and Disable Gate Drvier
output */
        if(GATEDRV_ResetREG() == ERROR)
        {
          printf("Reset GATEDRV Failed!\n");
          while(1);
        }

        /* Select RCO0 as system clock */
        CLOCK->GLBCLKCTL.bit.CLK0SEL = GLBCLKCTL_BIT_CLK0SEL_RCO0;
        CLOCK->GLBCLKCTL.bit.CLK1SEL = GLBCLKCTL_BIT_CLK1SEL_RCO1;

        /* Update Global variable */
        SysInfo.u32CLK0 = 24000000;
        SysInfo.u32CLK1 = SysInfo.u32CLK0;

        /* Close XO clock */
        CLOCK->XOCTL.bit.EN = 0;

        /* Close PLL clock */
        CLOCK->PLLCTL0.bit.EN = 0;

        /* Close Peripherals */
        CLOCK_DisableModule(SSP_MODULE);
        CLOCK_DisableModule(QSPI_MODULE);
        CLOCK_DisableModule(I2C_MODULE);
        CLOCK_DisableModule(ADC_MODULE);
        CLOCK_DisableModule(PWM0_MODULE);
        CLOCK_DisableModule(ECAP_MODULE);
        CLOCK_DisableModule(PWM1_MODULE);
        CLOCK_DisableModule(PWM2_MODULE);
        CLOCK_DisableModule(PWM3_MODULE);
        CLOCK_DisableModule(PWM4_MODULE);
        CLOCK_DisableModule(PWM5_MODULE);
        CLOCK_DisableModule(PWM6_MODULE);
        CLOCK_DisableModule(WDT0_MODULE);
        CLOCK_DisableModule(WDT1_MODULE);

        /* Low CPU and Bus clock frequency */
        CLOCK->PCLKCTL.bit.DIV    = 63;
        CLOCK->HCLKCTL.bit.DIV    = 63;
        /* Low UART clock frequency */
        CLOCK->UARTCLKCTL.bit.DIV = 63;


        /* Update system variable */
        SysInfo.u32HCLK = SysInfo.u32CLK0 / 63;
```

```
        SysInfo.u32PCLK = SysInfo.u32CLK0 / 63;
        SystemCoreClock = SysInfo.u32HCLK;

        /* Re-Init delay */
        Delay_Init();

        /* Re-Init UART */
        UART_Init(UART, 1200);
        printf("Enter Low Power Mode!\n");

        break;

      case 'W':  /* WAKE */

        /* Clock Init */
        CLOCK_InitWithRCO(CLOCK_HCLK_24MHZ);

        /* Delay Init */
        Delay_Init();


        /* Enable the peripherals you wanted here */
        CLOCK_EnableModule(QSPI_MODULE);
        CLOCK_EnableModule(ADC_MODULE);


        /* UART Init */
        UART_Init(UART, 1200);

        /* Gate Driver Init */
        if(GATEDRV_Init(1000000, &u8NewtonID, 1000) == ERROR)
        {
          printf("GATEDRV Init Failed!\n");
          while(1);
        }

        /* Reset GATEDRV */
        if(GATEDRV_ResetREG() == ERROR)
        {
          printf("Reset GATEDRV Failed!\n");
          while(1);
        }

        /* Set GATEDRV output enable */
        GATEDRV_EnablePreDriverOutput();
        GATEDRV_EnableGate();

        printf("Exit Low Power Mode!\n");

        break;

      default:
```

```
            printf("Invalid Command!\n");
            break;
        }
    }
}
```

经过实验测试，按照上述代码进入低功耗模式后，SPD1078 消耗的电流约为 2.18mA，功耗为 16V x 2.18mA = 34.88mW。下图为 UART 交互的结果。

图 1-1：低功耗实验结果