# SCM1612
# Wi-Fi 6 和 BLE 5 低功耗 SoC

# SDK 外设示例

文档版本 1.0
发布日期 2023-08-15



联系方式
速通半导体科技有限公司 (www.senscomm.com)
江苏省苏州市工业园区苏州大道西 2 号国际大厦 303 室
销售或技术支持，请发送电子邮件至
support@senscomm.com

## 免责声明和注意事项

本文档仅按"现状"提供。速通半导体有限公司保留在无需另行通知的情况下对其或本文档中包含的任何规格进行更正、改进和其他变更的权利。

与使用本文档中的信息有关的一切责任，包括侵犯任何专有权利的责任，均不予承认。此处不授予任何明示或暗示、通过禁止或其他方式对任何知识产权的许可。

本文档中的所有第三方信息均按"现状"提供，不对其真实性和准确性提供任何保证。

本文档中提及的所有商标、商号和注册商标均为其各自所有者的财产，特此确认。

# 版本历史

| 版本 | 日期 | 描述 |
| --- | --- | --- |
| 1.0 | 2024-03-05 | 初稿 |
| | | |
| | | |
| | | |
| | | |

# 目录

# 1 引言

## 1.1 示例

本文档描述了如何构建和运行 SDK 中提供的示例应用程序。

SDK 包含各种类型的示例应用程序。要探索可用的示例，请导航至 **[SDK]/api/examples** 目录。

目录结构如下：

```
api/examples/
├── peripherals
│   ├── adc
│   ├── i2c_eeprom
│   ├── ledc
│   └── spi_transfer
├── power_save
├── protocols
│   ├── common
│   ├── http_server
│   │   └── simple
│   └── mqtt
└── wifi
    ├── cli
    ├── softap
    └── sta
```

参考 **SCM1612 SDK 入门指南** 准备构建环境。

## 1.2 构建和配置

要构建特定的示例应用程序，请按照以下步骤操作：

- 从 SDK 的根目录，使用以下命令配置基本选项：

```
$ make scm1612s_defconfig
```

- 要更改配置或选择示例应用程序，请使用以下命令：

```
$ make menuconfig
```

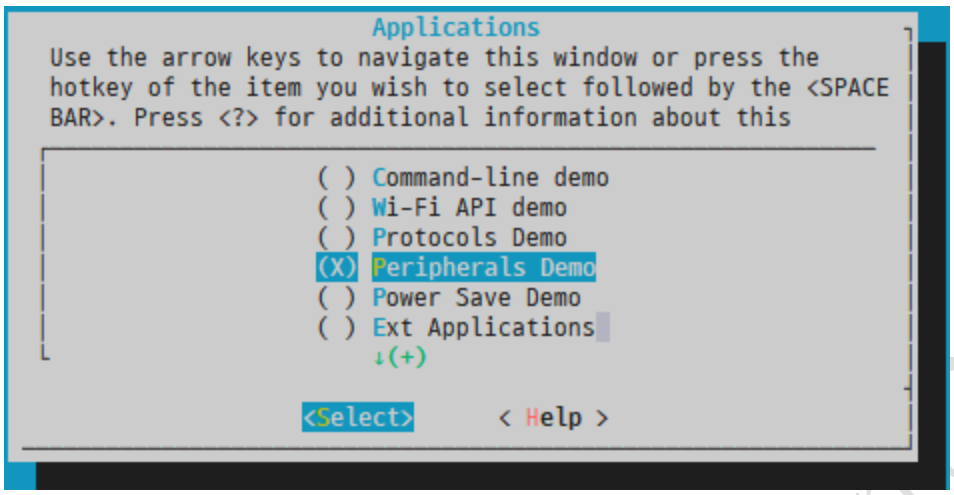
- 导航至 **Applications --->**

```
    Target platform  --->
    Kernel  --->
    Libraries/middleware  --->
[*] Enable WISE debug confgiurations  ----
-*- Command line interface  --->
[ ] AT commands  ----
[ ] Smart Configuration  ----
-*- Tinycrypt
[*] BLE library  --->
[ ] TinyUSB USB stack  ----
[*] MCUBoot  --->
[*] SDK  --->
-*- wise API  --->
    Applications  --->
```

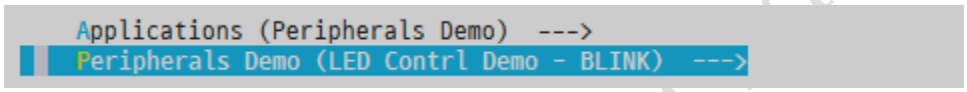
- 选择 **Applications (Command-line demo) --->**

```
    Applications (Command-line demo)  --->
```

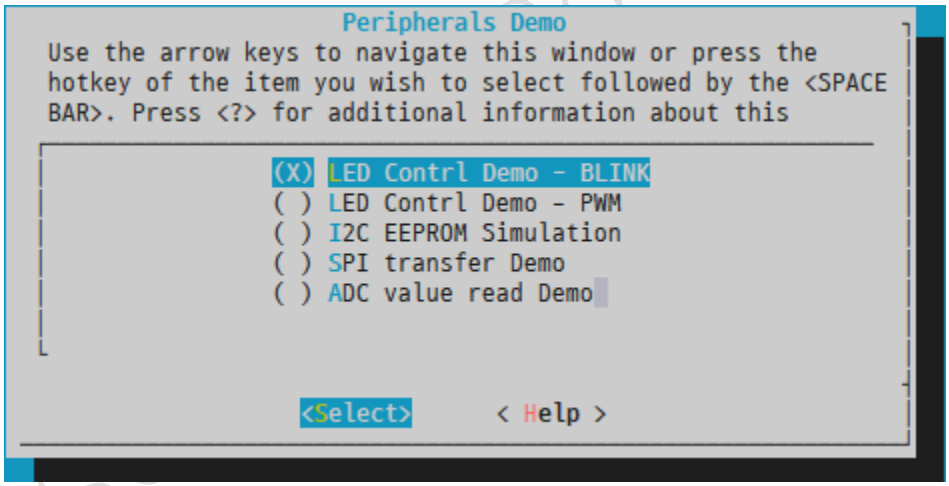
- 选择所需的示例。




- 如果有多个应用程序，将出现一个子菜单。




- 从子菜单中选择所需的示例。




- 使用以下命令构建固件：

```
$ make
```

构建成功完成后，将生成 `wise.mcuboot.bin` 文件，可以加载到开发板上。

## 1.3 应用程序入口点

名为 "init" 的第一个 OS 线程在线程上下文中调用 `main()` 函数。

Wise SDK 将默认 `main()` 定义为一个弱函数，它什么都不做，只返回 0。如果应用程序定义了一个 `main()` 函数，那么将调用它。例如，`api/examples/peripherals/i2c_eeprom/main.c` 定义了主函数，它是 I2C 示例的主入口点。如果配置了 I2C 示例，固件将在启动后运行此函数。

```
int main(void)
{
    printf("I2C demo\n");

    eeprom_master_init();
    eeprom_slave_init();

    return 0;
}
```

一些示例有自己的主函数，而另一些则没有。在任何一种情况下，大多数示例都依赖于用户命令进行测试。

当使用控制台命令行接口 (CLI) 时，"init" 线程还处理命令输入并执行相应的函数。因此，`main()` 函数不应阻塞并应退出以允许处理命令输入。

## 1.4 重建

尝试不同的示例时，应注意不要引起 PINMUX 设置的冲突。建议通过运行清理命令从干净的状态开始配置。

```
$ make distclean
```

# 2 外设示例

## 2.1 LED 控制 – 闪烁

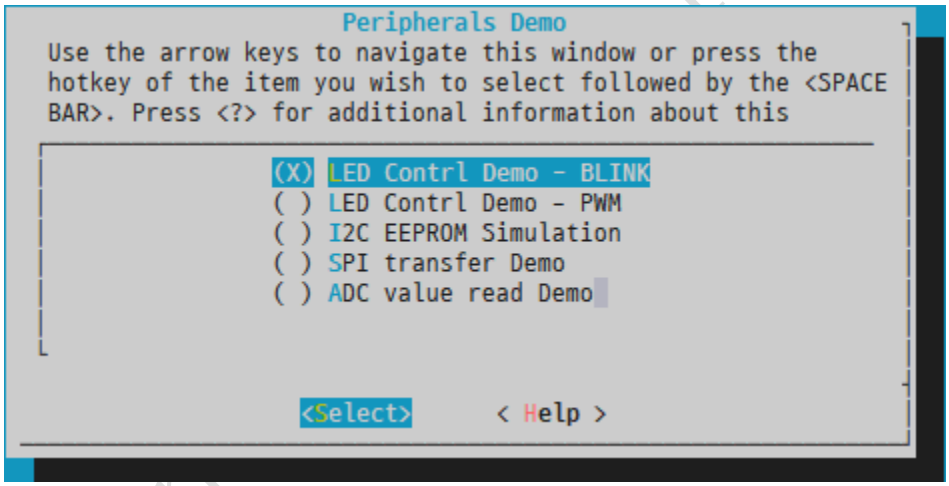
此示例展示了如何控制和切换 GPIO，使连接的 LED 能够开启或关闭。

[板子设置]

- 将 LED 连接到 GPIO16

[构建配置]

- 该示例可以从应用程序菜单中选择。




[示例]

- 启动后，GPIO 被配置为输出。
- 使用 `osDelay`，GPIO 在持续时间内切换。
- 过程重复配置的次数。

```c
void ledc_blink(void)
{
    /* configure GPIOs as outputs */
    scm_gpio_configure(GPIO_15, SCM_GPIO_PROP_OUTPUT);
    scm_gpio_configure(GPIO_23, SCM_GPIO_PROP_OUTPUT);
    scm_gpio_configure(GPIO_24, SCM_GPIO_PROP_OUTPUT);
    scm_gpio_configure(GPIO_RGB, SCM_GPIO_PROP_OUTPUT);

    /* start toggling */
    gpio_toggle_blinking(GPIO_RGB, true);

    /* set high */
    scm_gpio_write(GPIO_RGB, SCM_GPIO_HIGH_TO_LOW);
}

int main(void)
{
    printf("LEDC Blink demo\n");

    ledc_blink();

    return 0;
}
int demo_ledctrl_blink(int argc, char *argv[])
{
    /* configure GPIOs as outputs */
    scm_gpio_configure(GPIO_15, SCM_GPIO_PROP_OUTPUT);
    scm_gpio_configure(GPIO_23, SCM_GPIO_PROP_OUTPUT);
    scm_gpio_configure(GPIO_24, SCM_GPIO_PROP_OUTPUT);
    scm_gpio_configure(GPIO_RGB, SCM_GPIO_PROP_OUTPUT);

    /* start toggling */
    gpio_toggle_blinking(GPIO_RGB, true);

    /* set high */
    scm_gpio_write(GPIO_RGB, SCM_GPIO_LOW_TO_HIGH);

    return 0;
}
```

执行此示例时，控制台日志如下所示。

```
WISE 2018.02+ (Mar 11 2024 - 12:14:24 +0900)
LEDC Blink demo
I (659) DEMO_LEDCTRL: cnt = 0
I (1259) DEMO_LEDCTRL: cnt = 1
I (1859) DEMO_LEDCTRL: cnt = 2
I (2459) DEMO_LEDCTRL: cnt = 3
I (3059) DEMO_LEDCTRL: cnt = 4
I (3659) DEMO_LEDCTRL: cnt = 5
I (4259) DEMO_LEDCTRL: cnt = 6
I (4859) DEMO_LEDCTRL: cnt = 7
I (5459) DEMO_LEDCTRL: cnt = 8
I (6059) DEMO_LEDCTRL: cnt = 9
I (6659) DEMO_LEDCTRL: cnt = 10
I (7259) DEMO_LEDCTRL: cnt = 11
I (7859) DEMO_LEDCTRL: cnt = 12
I (8459) DEMO_LEDCTRL: cnt = 13
I (9059) DEMO_LEDCTRL: cnt = 14
$
```

## 2.2 LED 控制 - PWM

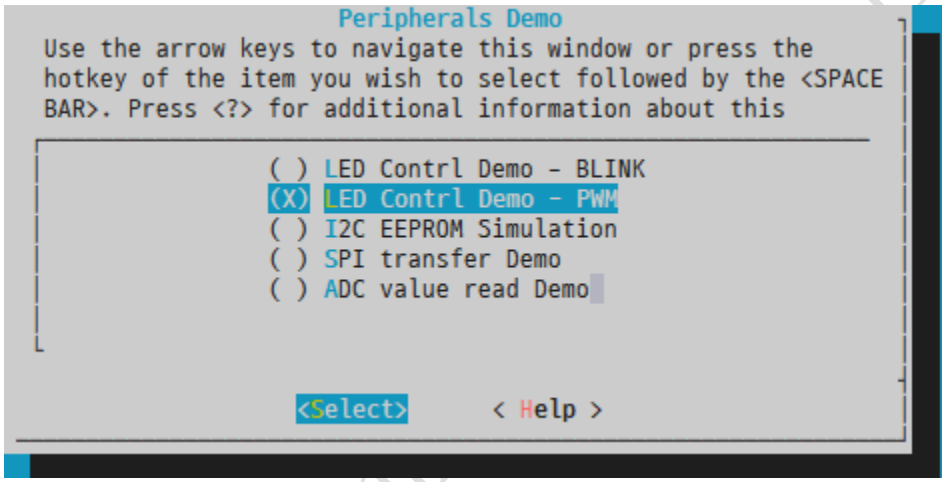
此示例展示了如何控制定时器的 PWM 功能以控制 LED。

[板子设置]

- 将 LED 连接到 GPIO15

[构建配置]

- 可以从应用程序菜单中选择该示例。

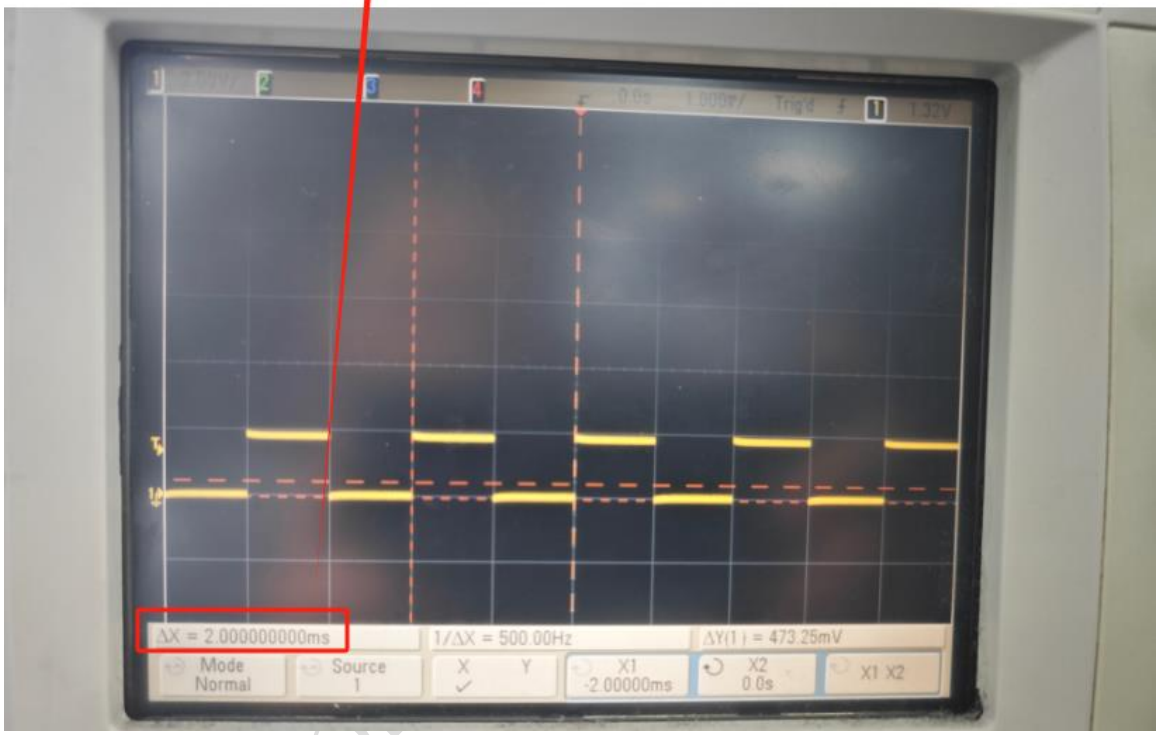



[示例]

```c
void ledc_pwm(void)
{
    struct scm_timer_cfg cfg;

    /* you can further adjust PWM parameters to change LED brightness.
     * for example, by modifying the high and low values through the corresponding function calls
     */

    /* Configure as PWM mode */
    cfg.mode = SCM_TIMER_MODE_PWM;
    cfg.intr_en = 0;            /* Disable interrupts */
    cfg.data.pwm.high = 1000;   /* Duration of high level (unit: microseconds) */
    cfg.data.pwm.low = 1000;    /* Duration of low level (unit: microseconds) */
    cfg.data.pwm.park = 0;      /* Park value, set to 0 */

    /* Call the TIMER driver configuration function */
    int ret = scm_timer_configure(LED_TIMER_IDX, LED_TIMER_CH, &cfg, NULL, NULL);
    if (ret) {
        printf("TIMER PWM configure error = %x\n", ret);
    } else {
        /* Start the TIMER */
        scm_timer_start(LED_TIMER_IDX, LED_TIMER_CH);
    }
}

int main(void)
{
    printf("LEDC PWM demo\n");

    ledc_pwm();

    return 0;
}
```

启动后，根据配置的持续时间生成 PWM 信号。

```
/* Configure as PWM mode */
cfg.mode = SCM_TIMER_MODE_PWM;
cfg.intr_en = 0;                        /* Disable interrupts */
cfg.data.pwm.high = 1000;               /* Duration of high level (unit: microseconds) */
cfg.data.pwm.low = 1000;                /* Duration of low level (unit: microseconds) */
cfg.data.pwm.park = 0;                  /* Park value, set to 0 */
```



## 2.3 I2C

此示例展示了 I2C 主机和从机的特性。

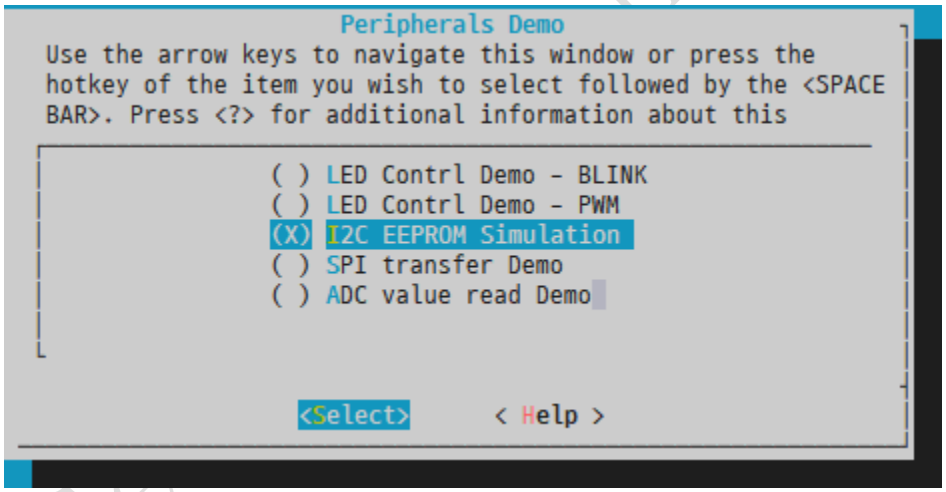
此示例中使用了两个 I2C。I2C0 用作主机，I2C1 用作从机。此示例模拟了典型的 EEPROM。

[板子设置]

- 要运行此示例，必须使用跳线设置板子。进行以下连接：
  - GPIO15 和 GPIO17
  - GPIO16 和 GPIO18

[构建配置]

- 可以从应用程序菜单中选择该示例。




[示例]

- 示例代码实现了初始化 I2C 主机和从机。一旦它们都初始化了，就可以接受用户命令并执行所需的操作。

```c
int main(void)
{
    printf("I2C demo\n");

    eeprom_master_init();
    eeprom_slave_init();

    return 0;
}
```

```c
int eeprom_master_init(void)
{
    struct scm_i2c_cfg cfg;
    int ret;

    memset(&cfg, 0, sizeof(cfg));
    cfg.role = SCM_I2C_ROLE_MASTER;
    cfg.master_clock = 100 * 1000;
    cfg.dma_en = 1;
    cfg.pull_up_en = 1;

    ret = scm_i2c_init(EEPROM_I2C_MASTER_IDX);
    if (ret) {
        printf("i2c init error %x\n", ret);
        return ret;
    }

    ret = scm_i2c_configure(EEPROM_I2C_MASTER_IDX, &cfg, eeprom_master_notify, NULL);
    if (ret) {
        printf("i2c configure error %x\n", ret);
        return ret;
    }

    return 0;
}
```

```c
int eeprom_slave_init(void)
{
    struct scm_i2c_cfg cfg;
    int ret;

    memset(&cfg, 0, sizeof(cfg));
    cfg.role = SCM_I2C_ROLE_SLAVE;
    cfg.dma_en = 0;
    cfg.pull_up_en = 1;
    cfg.slave_addr = EEPROM_DEVICE_ADDR;

    ret = scm_i2c_init(EEPROM_I2C_SLAVE_IDX);
    if (ret) {
        printf("i2c init error %x\n", ret);
        return ret;
    }

    ret = scm_i2c_configure(EEPROM_I2C_SLAVE_IDX, &cfg, eeprom_slave_notify, NULL);
    if (ret) {
        printf("i2c configure error %x\n", ret);
        return ret;
    }

    eeprom.offset = 0;
    for (int i = 0; i < EEPROM_MEMORY_SIZE; i++) {
        eeprom.data[i] = i;
    }

    return 0;
}
```

- 启动后，使用 "eeprom" 命令进行测试。示例控制台日志如下所示。用户可以尝试在不同的 EEPROM 地址处写入和读取不同长度的数据。

```
$ eeprom read

000: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
010: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
020: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
030: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
040: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
050: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
060: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
070: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
080: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
090: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0a0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0b0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0c0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0d0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0e0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0f0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
100: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
110: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
120: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
130: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
$ eeprom write 0x80 abcdefgh
$ eeprom read

000: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
010: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
020: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
030: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
040: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
050: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
060: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
070: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
080: 61 62 63 64 65 66 67 68 ff ff ff ff ff ff ff ff
090: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0a0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0b0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0c0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0d0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0e0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0f0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
100: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
110: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
120: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
130: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
```

## 2.4 SPI

此示例展示了通用的 SPI 传输。

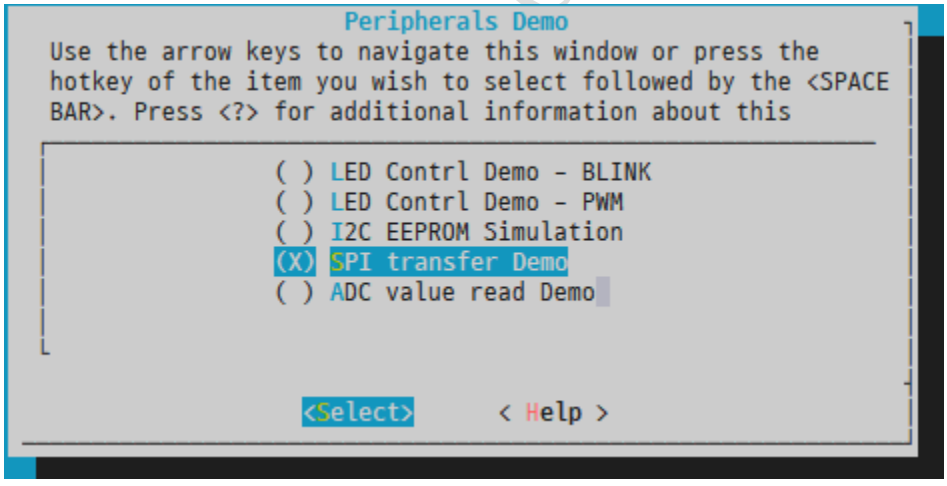
此示例中使用了两个 SPI。SPI1 用作主机，SPI2 用作从机。

[板子设置]

- 要运行此示例，必须使用跳线设置板子。进行以下连接：
  - GPIO15 和 GPIO2
  - GPIO16 和 GPIO3
  - GPIO17 和 GPIO4
  - GPIO18 和 GPIO5
  - GPIO19 和 GPIO6
  - GPIO20 和 GPIO7

[构建配置]

- 可以从应用程序菜单中选择该示例。




[示例]

- 使用 "spi_tr" 命令，可以测试带有单 IO、双 IO 或四 IO 的 SPI 传输。该命令将从主设备传输预定义的消息到从设备。

```
$ spi_tr 0 1
Start SPI Single IO Test
[SL->MS] SCM SPI Msg #0
[MS->SL] SCM SPI Msg #0
$ spi_tr 1 1
Start SPI Dual IO Test
[SL->MS] SCM SPI Msg #0
[MS->SL] SCM SPI Msg #0
$ spi_tr 2 1
Start SPI Quad IO Test
[SL->MS] SCM SPI Msg #0
[MS->SL] SCM SPI Msg #0
```

## 2.5 ADC

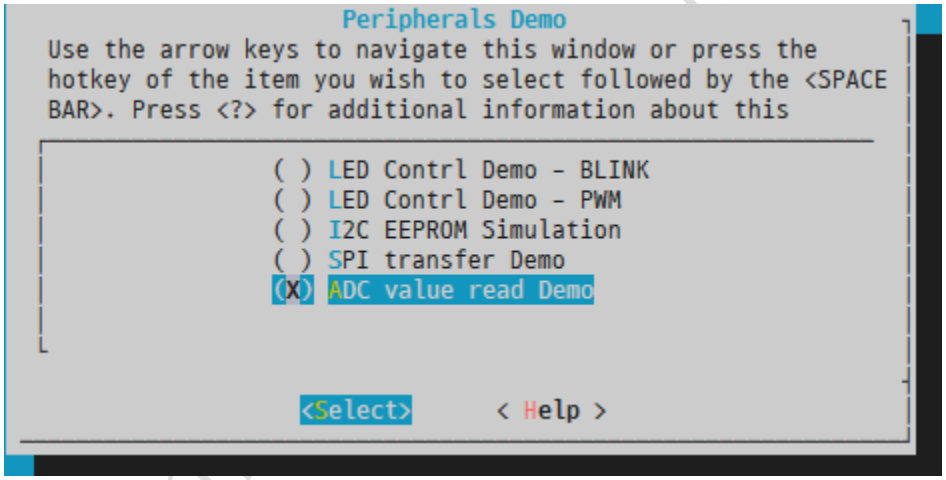
此示例展示了如何从 GPIO 引脚读取 ADC 值。

[板子配置]

- 将不同电平的信号连接到 GPIO：
  - 通道 4: GPIO4
  - 通道 5: GPIO7
  - 通道 6: GPIO0
  - 通道 7: GPIO1

[构建配置]

- 可以从应用程序菜单中选择该示例。




[示例]

- 使用 "adc" 命令读取一个通道所需的次数。更改输入电平后再次尝试。

```
$ adc read 5 8
ADC channel 5
0ccb 0c2c 0be9 0aaf 0964 082c 0704 05e9
$ adc read 5 8
ADC channel 5
0001 0000 0000 0000 0000 0000 0000 0000
```