SCM1612
Wi-Fi 6 and BLE 5 Low-Power SoC

SDK Demo Peripheral

Revision 1.0
Date 2023-08-15

Contact Information
Senscomm Semiconductor (www.senscomm.com)
Room 303, International Building, West 2 Suzhou Avenue,
SIP, Suzhou, China
For sales or technical support, please send email to
info@senscomm.com

Disclaimer and Notice

This document is provided on an “as-is” basis only. Senscomm reserves the right to make corrections, improvements and other changes to it or any specification contained herein without further notice.
All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.
All third party’s information in this document is provided as is with NO warranties to its authenticity and accuracy.
All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners and are hereby acknowledged.

Version History

Version	Date	Description
1.0	2024-03-05	Initial Draft

Table of Contents

1 Introduction

1.1 Demos

This document describes how to build and run demo applications provided within the SDK.

The SDK includes various types of demo applications. To explore the available examples, navigate to the • [SDK]/api/examples • directory.

The directory structure is as follows:

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

Refer to `SCM1612 SDK Getting Started Guide` to prepare the build environment.

1.2 Build & Configuration

To build a specific demo application, follow the procedures below:

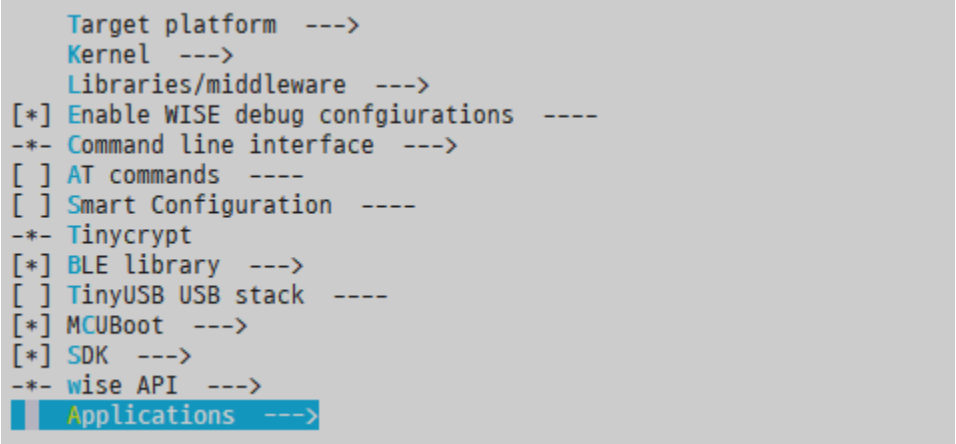
- From the root directory of the SDK, use the following command to configure the basic options.

```
$ make scm1612s_defconfig
```

- To change the configuration or select a demo application, use the following command:

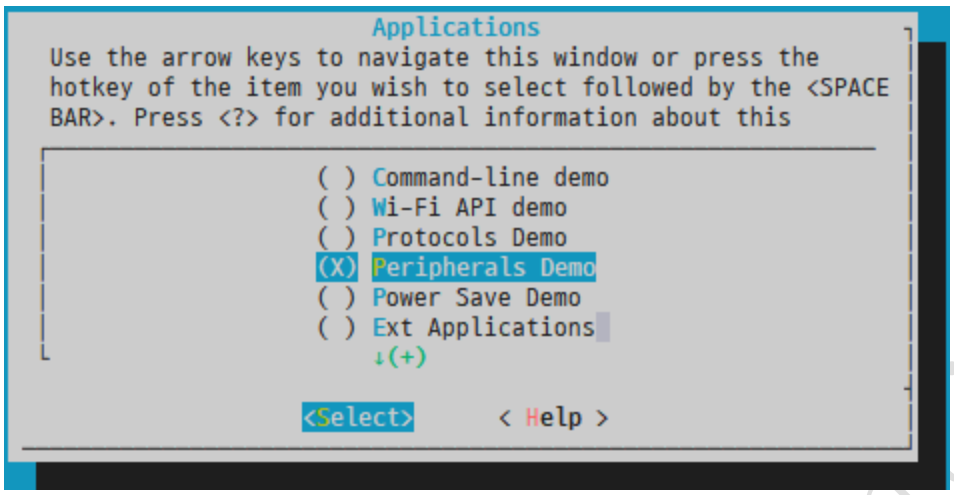
```
$ make menuconfig
```

- Navigate to `Applications --->`

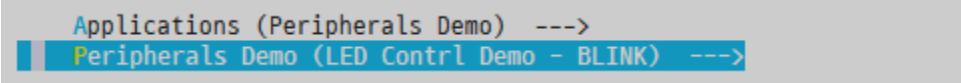


- Select `Applications (Command-line demo) --->`

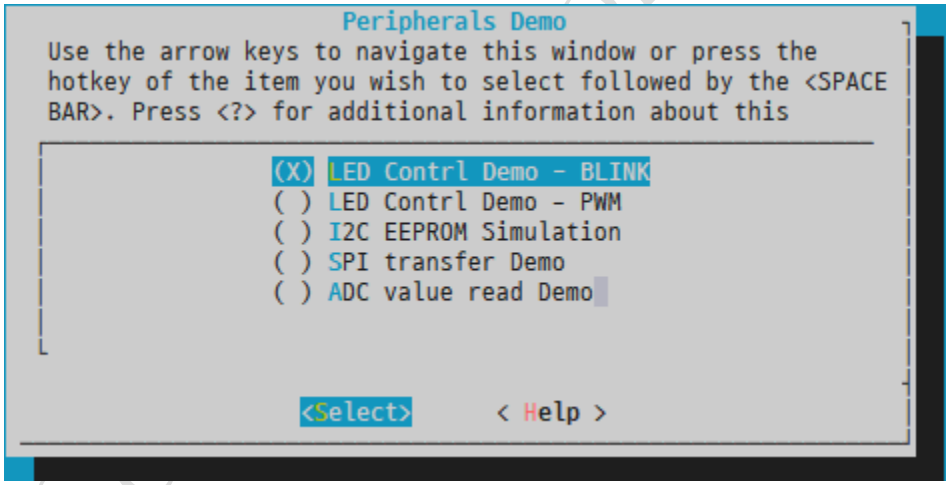
- Choose the desired demo.



- If there are multiple applications, a submenu will appear.



- Select the desired demo from the submenu.



- Use the following command to build the firmware.

```
$ make
```

When the build finishes successfully, the `wise.mcuboot.bin` file will be generated, which can be loaded onto a board.

1.3 Application Entry Point

The first OS thread, named "init" thread, calls the `main()` function in a thread context.

Wise SDK define default `main()` as a weak function, which does nothing but returns 0. If an application defines a `main()` function, then it will be invoked instead. For example, `api/examples/peripherals/i2c_eeprom/main.c` defines the main function, which it is the i2c demo's main entry point. If the I2C demo is configured, the firmware will run this function after the boot.

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

    eeprom_master_init();
    eeprom_slave_init();

    return 0;
}
```

Some demos have their own main functions while others do not have one. In either case, most of the demos rely on user commands for testing purposes.

When using the console command-line interface (CLI), the "init" thread also handles the command input and executes the corresponding functions. Therefore, the `main()` function should not block and should exit to allow processing the command inputs.

1.4 Rebuild

When trying a different demo, care must be taken not to cause a conflict on the PINMUX settings. It is recommeded to start the configuration from a clean state by running the clean command.

```
$ make distclean
```

2 Peripheral Demo

2.1 LED Control - Blink

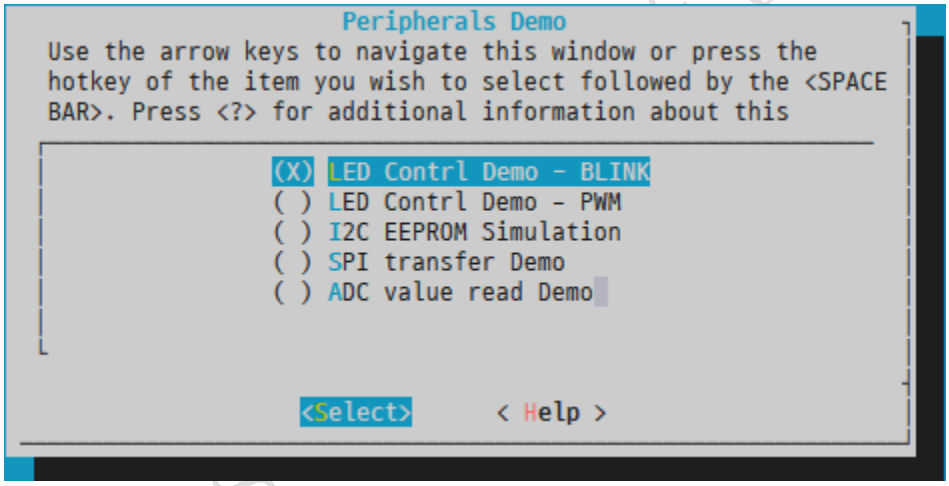
This demo shows how to control and toggle GPIOs so that the connected LED can be turned on or off.

[Board setup]

- Connect LED to GPIO16

[Build Configuration]

- The demo can be selected from the application menu.



[Demo]

- After boot, GPIOs are configured to be output.
- Using `osDelay`, GPIOs are toggled for the duration
- The process repeats configured number of times.

```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;
}
```

When executing this demo, the console log shows as below.

```
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 Control - PWM

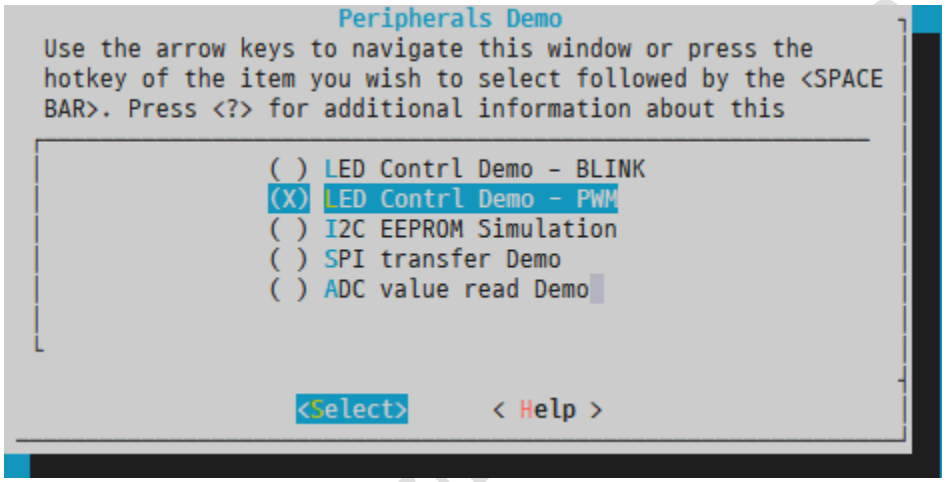
This demo shows how to control TIMER's PWM feature to control LEDs.

[Board Setup]

- Connect LED to GPIO15

[Build Configuration]

- The demo can be selected from the application menu.

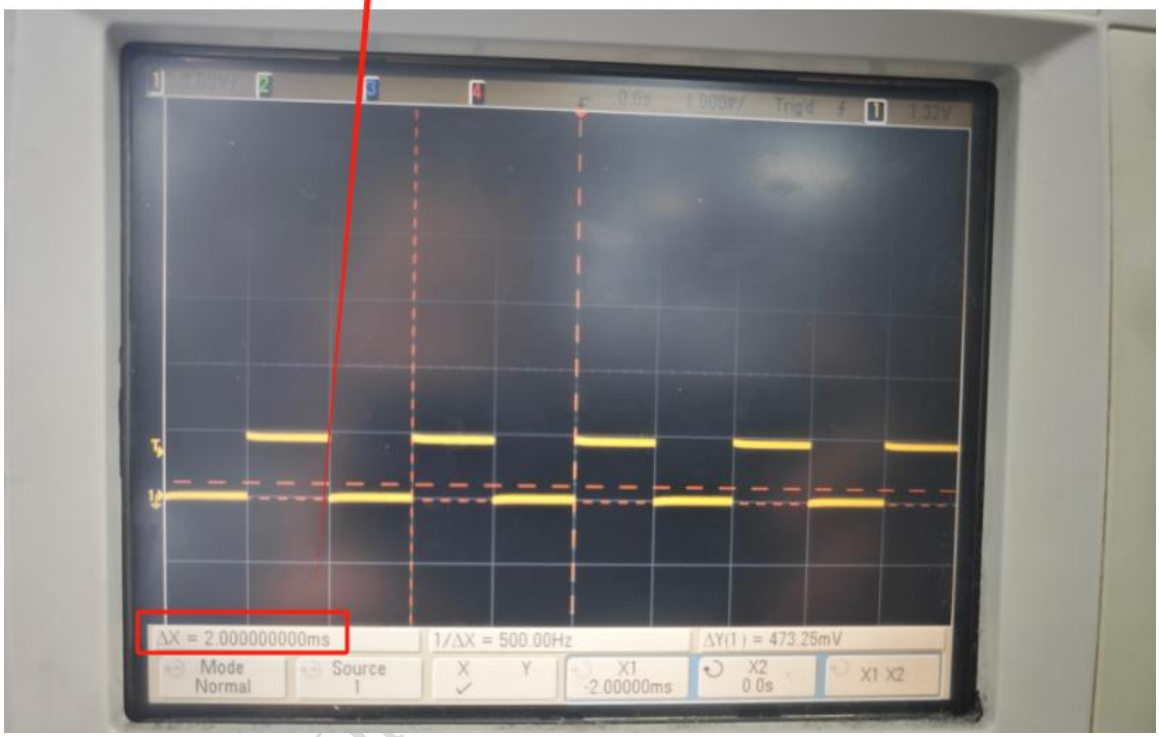


[Demo]

```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;
}
```

After boot, the PWM signal is generated according to the configured duration.

```
/* 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

This demo shows I2C master and slave features.

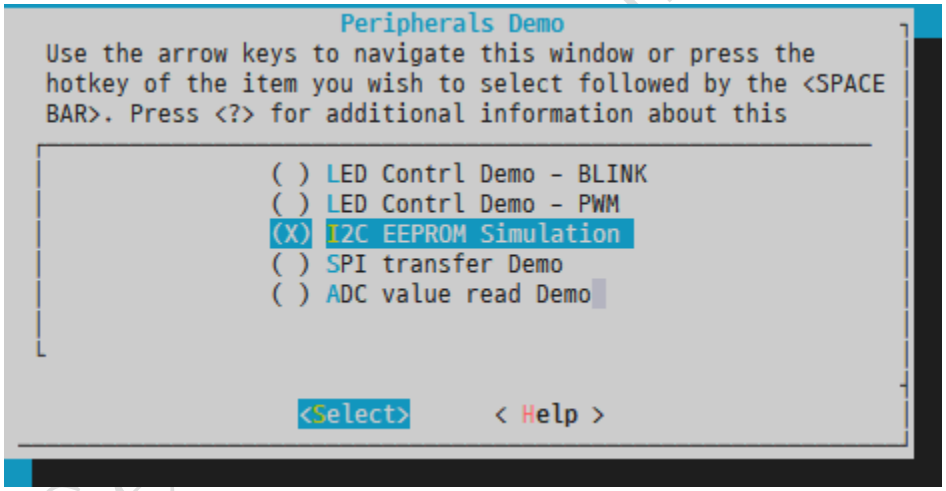
Two I2Cs are used in this demo. I2C0 is used as a master, and I2C1 is used as a slave.
The demo simulates a typical EEPROM.

[Board Setup]

- To run this demo, the board must be set up using jump wires. Make the following connections.
  - GPIO15 and GPIO17
  - GPIO16 and GPIO18

[Build Configuration]

- The demo can be selected from the application menu.



[Demo]

- The demo code implements initializing the I2C master and slave. Once they are both initialized, it is ready to accept user commands and perform the required operations.

```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;
}
```

- After boot, test with "eeprom" commands. Example console log shows as follows. User can try writing and reading data of variable length at different EEPOM address.

```
$ 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

This demo shows a general SPI transfer.

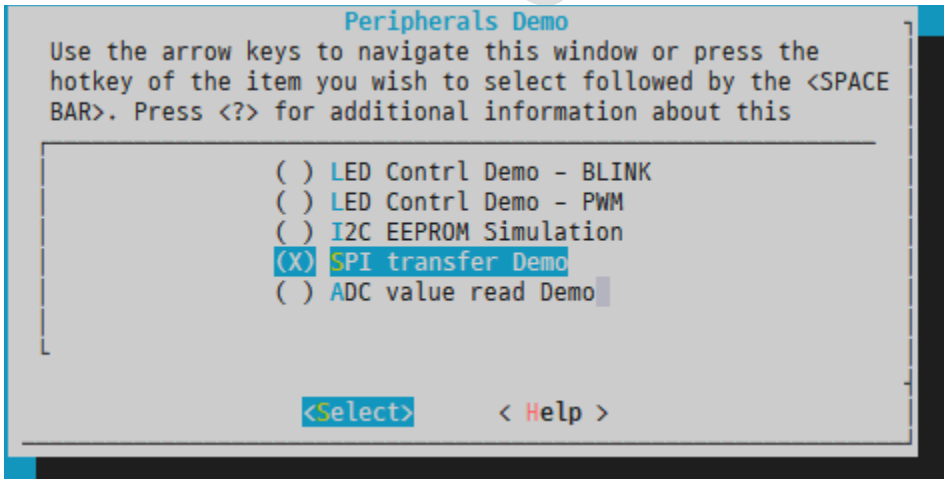
Two SPIs are used in this demo. SPI1 is used as a master, SPI2 is used as a slave.

[Board Setup]

- To run this demo, the board must be set up using jump wires. Make the following connections.
  - GPIO15 and GPIO2
  - GPIO16 and GPIO3
  - GPIO17 and GPIO4
  - GPIO18 and GPIO5
  - GPIO19 and GPIO6
  - GPIO20 and GPIO7

[Build Configuration]

- The demo can be selected from the application menu.



[Demo]

- Using the "spi_tr" command, SPI transfer with signle IO, dual IO, or quad IO can be tested. The command will transfer pre-defined message from the master to the slave.

```
$ 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

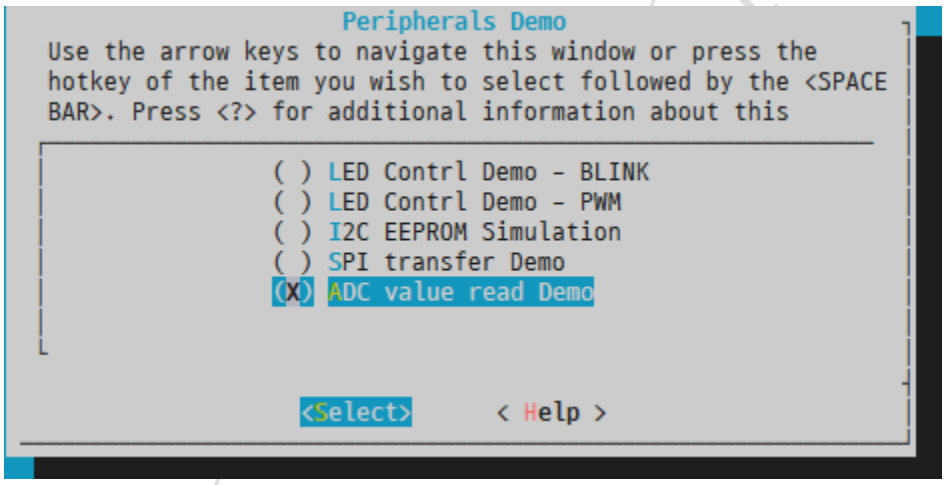
This demo shows reading an ADC value from a GPIO pin.

[Board Configuration]

- Connect a different level of signal to the GPIOs
  - Channel 4: GPIO4
  - Channel 5: GPIO7
  - Channel 6: GPIO0
  - Channel 7: GPIO1

[Build Configuration]

- The demo can be selected from the application menu.



[Demo]

- Use the "adc" command to read a channel for the required number of times. Try again after changing the input level.

```
$ 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
```