StickS3 Low-Power Configuration

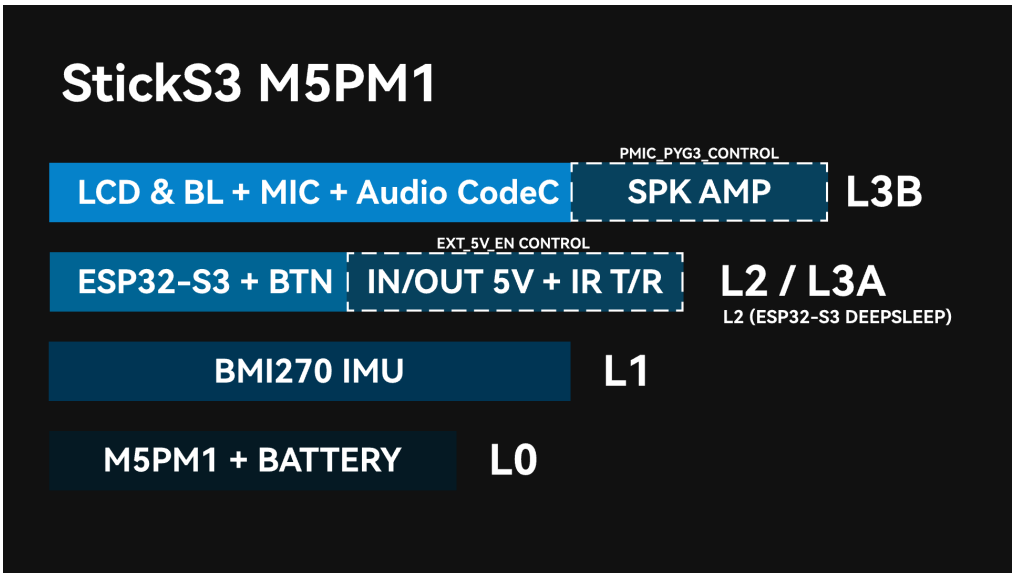
1. Multi-Level Power Switch Design

StickS3 integrates the M5PM1 internally. Combined with the hardware circuitry, it implements a multi-level power switch design. Different power switch levels control the power supply to corresponding peripherals and interfaces. Users can switch between different power enable levels according to runtime requirements, turning off unused peripheral sections to achieve low power consumption for the entire device.

M5PM1 Driver Library

Using the M5PM1 driver library allows convenient configuration of the M5PM1 pin functions, which are used for low-power wake-up and peripheral power switching.

- M5PM1 Arduino Library



Notes

The multi-level power switch design of StickS3 is not a cascaded structure. The power inputs of switches L1 ~ L3B all come from the L0 level source rather than the previous level. Therefore, independent control of different power switch levels is supported. The above levels are divided based on peripheral power supply and power consumption.

After the M5PM1 starts up and powers on, L0, L1, and L2 will be automatically enabled (by default enabling `DCDC3V3_EN_PP`, `LDO3V3_EN_PP`, and `CHG_EN_PP`). During the M5Unified initialization process, L3A and L3B will be further enabled to supply power to other peripherals.

L0

At this power level, the battery continues to supply power to the M5PM1, while power to other peripherals can be turned off. As long as the battery is not depleted, this power level will remain active. The M5PM1 supports basic power on/off operations via buttons at this level.

L1

Enables power supply to the IMU peripheral. The power switch at this level uses the M5PM1 `LDO3V3_EN_PP`, which can be controlled using the following API.

```
pm1.setLdoEnable(true); // L1 ON
pm1.setLdoEnable(false); // L1 OFF
```

At the same time, the IMU INT1 interrupt pin is connected to the M5PM1 PYG4. By configuring the relevant registers, IMU power can be maintained while the M5PM1 enters sleep mode. By flipping the device, the IMU interrupt can be triggered to wake up the M5PM1. The following API can be used to keep IMU (L1) power enabled.

```
pm1.setLdoEnable(true);
pm1.ldoSetPowerHold(true);
pm1.setLedEnLevel(true);
pm1.shutdown();
```

L2/L3A

At this power level, the ESP32-S3, Grove interface, Hat expansion interface, infrared transceiver, and button pull-up circuitry are powered.

When the ESP32-S3 is in sleep mode, the power level is L2. When the ESP32-S3 is in active operation, the power level is L3A.

The ESP32-S3 can turn off its own power supply (L2 -> L1/L0) by controlling the M5PM1 to enter sleep mode.

Among these, the power supply for the expansion interface input/output and the infrared circuit must be additionally controlled via the M5PM1 `EXT_5V_EN` (BOOST5V_EN_PP) pin.

Therefore, before connecting external sensors to the expansion interface or using the infrared transceiver function, please ensure that the peripherals are powered on.

> **EXT_5V_EN Input Power Notice**
>
> The device 5V power interface can be configured as DC 5V output/input mode. By default, the interface is set to input mode. In this mode, DC 5V power can be supplied via the Grove interface, the top Hat2-Bus EXT_5V, or the 5VIN interface. When configured as output mode, power input is only allowed via USB or the top Hat2-Bus 5VIN. Do not supply power from other output interfaces; otherwise, there is a risk of short circuit damage to the device.

In the default M5Unified initialization, EXT_5V_EN is disabled. This operation turns off the power supply to the Grove interface, Hat EXT_5V interface, and IR TX/RX, switching them to input mode. In this state, external 5V input power is required for IR TX/RX to operate properly. In scenarios without external power input, the following API can be used to re-enable EXT_5V output mode and restore power to IR TX/RX.

```
M5.Power.setExtOutput(true); // EXT_5V OUTPUT
// M5.Power.setExtOutput(false); // EXT_5V INPUT
```

L3B

This power level enables power supply to all peripherals. M5Unified initializes this power level by default, including the LCD backlight, MIC, and SPK. The power switch for this level uses the `PYG2` pin of M5PM1, which can be controlled via the following API.

```
pm1.gpioSetFunc(M5PM1_GPIO_NUM_2, M5PM1_GPIO_FUNC_GPIO);
pm1.gpioSetMode(M5PM1_GPIO_NUM_2, M5PM1_GPIO_MODE_OUTPUT);
pm1.gpioSetDrive(M5PM1_GPIO_NUM_2, M5PM1_GPIO_DRIVE_PUSHPULL);
pm1.gpioSetOutput(M5PM1_GPIO_NUM_2, false);
```

SPK AMP

The device SPK amplifier switch is controlled via the `PYG3` pin of M5PM1. This part can be enabled or disabled using either the M5Unified API or the M5PM1 API.

Note: When using the IR receive function, the SPK amplifier must be turned off.

```
M5.Speaker.begin(); // Initialize SPK and enable the amplifier
// M5.Speaker.end(); // Release SPK and disable the amplifier
```

or

```
pm1.gpioSetFunc(M5PM1_GPIO_NUM_3, M5PM1_GPIO_FUNC_GPIO);
pm1.gpioSetMode(M5PM1_GPIO_NUM_3, M5PM1_GPIO_MODE_OUTPUT);
pm1.gpioSetDrive(M5PM1_GPIO_NUM_3, M5PM1_GPIO_DRIVE_PUSHPULL);
pm1.gpioSetOutput(M5PM1_GPIO_NUM_3, true);  // Enable SPK amplifier
// pm1.gpioSetOutput(M5PM1_GPIO_NUM_3, false); // Disable SPK amplifier
```

2. M5PM1 Sleep

Manual Sleep

M5PM1 can be manually controlled by software to enter sleep mode to reduce overall system power consumption. By default, directly entering sleep will fall back to the L0 power level, where only M5PM1 remains powered.

```
pm1.shutdown();
```

In some special use cases (such as IMU wake-up or ESP32-S3 SoC sleep), M5PM1 is allowed to enter sleep mode while still keeping power supplied to certain levels of peripherals, which can be used as wake-up sources or for state retention.

Such applications require configuring the power switch pin states of the corresponding levels and enabling power hold before M5PM1 enters sleep mode.

I2C Idle Sleep

M5PM1 supports configuring automatic sleep based on I2C idle time to reduce overall system power consumption. After entering sleep mode, the first communication between ESP32-S3 and M5PM1 will be used to wake up M5PM1. Since this communication will fail, valid communication will occur on the next transaction after wake-up.

```
m5pm1_err_t setI2cSleepTime(uint8_t seconds);
```

3. M5PM1 Timer

M5PM1 supports timer configuration. When the timer expires, corresponding actions can be executed, such as power on, power off, or reboot.

```
m5pm1_err_t timerSet(uint32_t seconds, m5pm1_tim_action_t action);
```

```
typedef enum {
    M5PM1_TIM_ACTION_STOP = 0b000,     // Stop, no action
    M5PM1_TIM_ACTION_FLAG = 0b001,     // Set flag only
    M5PM1_TIM_ACTION_REBOOT = 0b010,   // System reboot
    M5PM1_TIM_ACTION_POWERON = 0b011,  // Power on
    M5PM1_TIM_ACTION_POWEROFF = 0b100  // Power off
} m5pm1_tim_action_t;
```

Timed Power On / Power Off Example

Example description: After the device powers on, a single click of Button A configures a 10-second timer to trigger a reboot. A single click of Button B configures a 10-second timer that triggers M5PM1 to power off (the device can be powered on again by pressing the power button).

cpp

```cpp
#include <M5Unified.h>
#include <M5PM1.h>
#include <Wire.h>

M5PM1 pm1;

void setup(void)
{
    M5.begin();
    M5.Display.setRotation(1);
    Serial.begin(115200);
    auto pin_num_sda = M5.getPin(m5::pin_name_t::in_i2c_sda);
    auto pin_num_scl = M5.getPin(m5::pin_name_t::in_i2c_scl);
    M5_LOGI("getPin: SDA:%u SCL:%u", pin_num_sda, pin_num_scl);
    Wire.end();
    Wire.begin(pin_num_sda, pin_num_scl, 100000U);

    // Initialize PM1
    m5pm1_err_t err = pm1.begin(&Wire, M5PM1_DEFAULT_ADDR, pin_num_sda, pin_num_scl, M5PM1_I2C_FREQ_

    if (err == M5PM1_OK) {
        Serial.println("PM1 initialization successful");
    } else {
        Serial.printf("PM1 initialization failed, error code: %d\n", err);
    }
    M5.Display.fillScreen(BLACK);
    M5.Display.setTextSize(2);
    M5.Display.setTextColor(WHITE);
    M5.Display.setCursor(0, 10);
    M5.Display.println("Timer Power Test");
    M5.Display.println("BtnA: After 10s ON");
    M5.Display.println("BtnB: After 10s OFF");
}

void loop(void)
{
    M5.update();
    if (M5.BtnA.wasPressed()) {
        M5.Display.fillScreen(BLACK);
        M5.Display.setCursor(0, 10);
        M5.Display.println("Shutdown");
        M5.Display.println("After 10s");
        M5.Display.println("Power ON");
        delay(1000);
        pm1.timerSet(10, M5PM1_TIM_ACTION_POWERON);
        pm1.shutdown();
    }
    if (M5.BtnB.wasPressed()) {
        M5.Display.fillScreen(BLACK);
        M5.Display.setCursor(0, 10);
```

```
        M5.Display.println("After 10s");
        M5.Display.println("Power OFF");
        delay(1000);
        pm1.timerSet(10, M5PM1_TIM_ACTION_POWEROFF);
    }
}
```

4. IMU Example Program

BMI270 Driver Library

- SparkFun_BMI270_Arduino_Library

IMU M5PM1 Wake-up

When the power is switched to **L1 Mode**, the entire device keeps only the IMU and M5PM1 powered. After configuring the IMU wake-up function, M5PM1 also enters sleep mode while maintaining the L1 output power (3V3_L1_EN) to keep the IMU operating.

At this point, flipping or moving the device can trigger the IMU wake-up signal, which wakes up M5PM1 and restarts the system.

After M5PM1 wakes up, it will re-execute the power-on sequence for L0, L1, and L2. The ESP32-S3 will then re-run its initialization process.

Example description: After the device powers on, single-click Button A to configure the IMU interrupt mode and enable M5PM1 L1 power hold, then M5PM1 enters sleep mode. Flipping or moving the device will trigger the IMU to wake up M5PM1, and the ESP32-S3 will be powered on again.

cpp

```cpp
#include <M5Unified.h>
#include <M5PM1.h>
#include <Wire.h>
#include "SparkFun_BMI270_Arduino_Library.h"

BMI270 imu;
M5PM1 pm1;

void setup(void)
{
    M5.begin();
    M5.Display.setRotation(1);
    Serial.begin(115200);
    auto pin_num_sda = M5.getPin(m5::pin_name_t::in_i2c_sda);
    auto pin_num_scl = M5.getPin(m5::pin_name_t::in_i2c_scl);
    M5_LOGI("getPin: SDA:%u SCL:%u", pin_num_sda, pin_num_scl);
    Wire.end();
    Wire.begin(pin_num_sda, pin_num_scl, 100000U);

    // Initialize PM1
    m5pm1_err_t err = pm1.begin(&Wire, M5PM1_DEFAULT_ADDR, pin_num_sda, pin_num_scl, M5PM1_I2C_FREQ_

    if (err == M5PM1_OK) {
        Serial.println("PM1 initialization successful");
        pm1.gpioSetWakeEnable(M5PM1_GPIO_NUM_4, true);
        pm1.gpioSetWakeEdge(M5PM1_GPIO_NUM_4, M5PM1_GPIO_WAKE_FALLING);  // Falling edge
    } else {
        Serial.printf("PM1 initialization failed, error code: %d\n", err);
    }

    // Check if sensor is connected and initialize
    // Address defaults to 0x68)
    while(imu.beginI2C(BMI2_I2C_PRIM_ADDR) != BMI2_OK)
    {
        Serial.println("Error: BMI270 not connected, check wiring and I2C address!");
        delay(1000);
    }
    imu.disableFeature(BMI2_ANY_MOTION);
    Serial.println("BMI270 initialization successful");

    M5.Display.setFont(&fonts::FreeMonoBold9pt7b);
    M5.Display.fillScreen(BLACK);
    M5.Display.setTextColor(WHITE);
    M5.Display.setCursor(0, 10);
    M5.Display.println("IMU Wakeup Test");
    M5.Display.println("Press BtnA to Sleep");
    M5.Display.println("Shake to wake up");
}

void loop(void)
```

```
{
    M5.update();
    if (M5.BtnA.wasPressed()) {
        int8_t ret = imu.enableFeature(BMI2_ANY_MOTION);
        // Optional
        // bmi2_sens_config config;
        // config.type = BMI2_ANY_MOTION;
        // config.cfg.any_motion.threshold = 0xA0;// 1LSB equals to 0.48mg. Default is 83mg. Lower i
        // config.cfg.any_motion.duration = 0x0A; // 1LSB equals 20ms. Default is 100ms.
        // ret |= imu.setConfig(config);
        //
        bmi2_int_pin_config intPinConfig;
        intPinConfig.pin_type = BMI2_INT1;
        intPinConfig.int_latch = BMI2_INT_NON_LATCH;
        intPinConfig.pin_cfg[0].lvl = BMI2_INT_ACTIVE_LOW;
        intPinConfig.pin_cfg[0].od = BMI2_INT_PUSH_PULL;
        intPinConfig.pin_cfg[0].output_en = BMI2_INT_OUTPUT_ENABLE;
        intPinConfig.pin_cfg[0].input_en = BMI2_INT_INPUT_DISABLE;
        ret |= imu.setInterruptPinConfig(intPinConfig);
        ret |= imu.mapInterruptToPin(BMI2_ANY_MOTION_INT, BMI2_INT1);
        if (!ret){
            Serial.println("BMI270 AnyMotionInterrupt enabled successfully");
        } else {
            Serial.println("Failed to enable BMI270 AnyMotionInterrupt");
        }

        M5.Display.fillScreen(BLACK);
        M5.Display.setCursor(0, 10);
        M5.Display.println("Power OFF");
        delay(1000);
        // Shutdown
        pm1.setLdoEnable(true);
        pm1.ldoSetPowerHold(true);
        pm1.setLedEnLevel(true);
        pm1.shutdown();
    }
}
```

Example Effect Demonstration:

StickS3_Arduino_imu_pmic.mp4

IMU ESP32-S3 Wake-up

The `PYG1_IRQ` pin of M5PM1 is electrically connected to `G13` of the ESP32-S3. By using a chained wake-up signal, the ESP32-S3 can be woken up. The implementation steps are as follows:

1. Configure M5PM1 `PYG4` as input mode. This pin is connected to the IMU `INT1` output signal.
2. Configure M5PM1 `PYG1_IRQ` as an IRQ output pin. When the state of `PYG4` (IMU interrupt signal) changes, the `PYG1_IRQ` pin will output a signal.
3. Configure the ESP32-S3 to enter sleep mode and set the wake-up pin to `G13`.
4. Flip or move the device: trigger the IMU wake-up signal → trigger M5PM1 `PYG1_IRQ` → wake up the ESP32-S3.

Example description: After the device powers on, click Button A to configure the IMU interrupt mode. Flipping or moving the device will trigger the GPIO interrupt handler. Click Button A again to clear the M5PM1 IRQ flag, and then perform the test again.

cpp

```cpp
#include <M5Unified.h>
#include <M5PM1.h>
#include <Wire.h>
#include "SparkFun_BMI270_Arduino_Library.h"
#include "driver/rtc_io.h"

BMI270 imu;
M5PM1 pm1;

void setup(void)
{
    M5.begin();
    M5.Display.setRotation(1);
    Serial.begin(115200);
    auto pin_num_sda = M5.getPin(m5::pin_name_t::in_i2c_sda);
    auto pin_num_scl = M5.getPin(m5::pin_name_t::in_i2c_scl);
    M5_LOGI("getPin: SDA:%u SCL:%u", pin_num_sda, pin_num_scl);
    Wire.end();
    Wire.begin(pin_num_sda, pin_num_scl, 100000U);

    // Initialize PM1
    m5pm1_err_t err = pm1.begin(&Wire, M5PM1_DEFAULT_ADDR, pin_num_sda, pin_num_scl, M5PM1_I2C_FREQ

    if (err == M5PM1_OK) {
        Serial.println("PM1 initialization successful");
        pm1.irqClearGpioAll();
        pm1.irqClearSysAll();
        pm1.irqClearBtnAll();

        pm1.irqSetGpioMaskAll(M5PM1_IRQ_MASK_ENABLE);
        pm1.irqSetSysMaskAll(M5PM1_IRQ_MASK_ENABLE);
        pm1.irqSetBtnMaskAll(M5PM1_IRQ_MASK_ENABLE);

        pm1.irqSetGpioMask(M5PM1_IRQ_GPIO4, M5PM1_IRQ_MASK_DISABLE);
        pm1.gpioSetMode(M5PM1_GPIO_NUM_4, M5PM1_GPIO_MODE_INPUT);
        pm1.gpioSetPull(M5PM1_GPIO_NUM_4, M5PM1_GPIO_PULL_UP);

        pm1.gpioSetMode(M5PM1_GPIO_NUM_1, M5PM1_GPIO_MODE_OUTPUT);
        pm1.gpioSetDrive(M5PM1_GPIO_NUM_1, M5PM1_GPIO_DRIVE_PUSHPULL);
        pm1.gpioSetFunc(M5PM1_GPIO_NUM_1, M5PM1_GPIO_FUNC_IRQ);
    } else {
        Serial.printf("PM1 initialization failed, error code: %d\n", err);
    }

    // Check if sensor is connected and initialize
    // Address is optional (defaults to 0x68)
    while(imu.beginI2C(BMI2_I2C_PRIM_ADDR) != BMI2_OK)
    {
        Serial.println("Error: BMI270 not connected, check wiring and I2C address!");
        delay(1000);
```

```
    }
    imu.disableFeature(BMI2_ANY_MOTION);
    Serial.println("BMI270 initialization successful");

    M5.Display.setFont(&fonts::FreeMonoBold9pt7b);
    M5.Display.fillScreen(BLACK);
    M5.Display.setTextColor(WHITE);
    M5.Display.setCursor(0, 10);
    M5.Display.println("IMU IRQ Test");
    M5.Display.println("Press BtnA to Sleep");
    M5.Display.println("Shake to wake up");
}

void ARDUINO_ISR_ATTR pm1_irq_handler()
{
    Serial.println("PM1 IRQ triggered");
    M5.Display.println("PM1 IRQ triggered");
}

void loop(void)
{
    M5.update();
    if (M5.BtnA.wasPressed()) {
        pm1.irqClearGpioAll();
        pm1.irqClearSysAll();
        pm1.irqClearBtnAll();
        int8_t ret = imu.enableFeature(BMI2_ANY_MOTION);
        // Optional
        // bmi2_sens_config config;
        // config.type = BMI2_ANY_MOTION;
        // config.cfg.any_motion.threshold = 0xE0;// 1LSB equals to 0.48mg. Default is 83mg. Lower
        // config.cfg.any_motion.duration = 0x0A; // 1LSB equals 20ms. Default is 100ms.
        // imu.setConfig(config);
        //
        bmi2_int_pin_config intPinConfig;
        intPinConfig.pin_type = BMI2_INT1;
        intPinConfig.int_latch = BMI2_INT_NON_LATCH;
        intPinConfig.pin_cfg[0].lvl = BMI2_INT_ACTIVE_LOW;
        intPinConfig.pin_cfg[0].od = BMI2_INT_PUSH_PULL;
        intPinConfig.pin_cfg[0].output_en = BMI2_INT_OUTPUT_ENABLE;
        intPinConfig.pin_cfg[0].input_en = BMI2_INT_INPUT_DISABLE;
        ret |= imu.setInterruptPinConfig(intPinConfig);
        ret |= imu.mapInterruptToPin(BMI2_ANY_MOTION_INT, BMI2_INT1);
        if (!ret){
            Serial.println("BMI270 AnyMotionInterrupt enabled successfully");
        } else {
            Serial.println("Failed to enable BMI270 AnyMotionInterrupt");
        }

        M5.Display.fillScreen(BLACK);
        M5.Display.setCursor(0, 10);
        M5.Display.println("Now Shake!");

```

```
        // Choose either of the two pieces of code below.
        pinMode(GPIO_NUM_13, INPUT_PULLUP);
        attachInterrupt(GPIO_NUM_13, pm1_irq_handler, FALLING);

        // esp_sleep_enable_ext0_wakeup(GPIO_NUM_13, 0);  // 0 = Low
        // rtc_gpio_pullup_en(GPIO_NUM_13);
        // Serial.println("Going to sleep now");
        // esp_deep_sleep_start();
    }
}
```

Example Demonstration:

StickS3_Arduino_imu_esp32s3.mp4