User's Guide

LP581X RUKA Sample Code

ABSTRACT

This document describes the preparation and usage of the sample code for the LP5815 device when paired with a LP-MSPM0L1306 device. The installed code lights up the light emitting diodes (LED) on the LP5815YCHEVM evaluation module when the setup instructions provided by TI are followed.

The LP5815 device is a three-channel inter-integrated circuit (I2C) interface red, green, and blue (RGB) LED driver with instant blinking and autonomous animation control. The sample code can be modified to control the LP5814, LP5814I, LP5816 and LP5817 family of devices.

Table of Contents

1 Introduction....2
2 Hardware Setup....3
3 Software Setup....4
4 Sample Code Structure....6
4.1 Design Requirements....6
4.2 Flow Diagram....6
4.3 Pattern Parameters....8
5 Revision History....11

List of Figures

Figure 2-1. Hardware Connection....3
Figure 3-1. Installation Process of Code Composer Studio....4
Figure 3-2. Verification of MSPM0 SDK Installation....5
Figure 4-1. Sample Code Flow Diagram....8
Figure 4-2. Pattern Example 1 Timing and Parameters....9
Figure 4-3. Pattern Example 2 Timing and Parameters....9
Figure 4-4. Pattern Example 3 Timing and Parameters....11

List of Tables

Table 4-1. LP5815 Design Parameters....6
Table 4-2. Engine Busy Lock Registers....7

Trademarks

1 Introduction

The sample code showcases how to configure the LP5815 device to turn on the RGB LED on the LP5815YCHEVM evaluation module with the LP-MSPM0L1306 evaluation board using the I2C communication.

There are three pattern examples demonstrated in the code:

- LEDs on and off in manual mode
- LEDs fade in and out (breathe) in manual mode
- LEDs breathe to flash in autonomous animation mode

All three pattern examples can be implemented on LP5814, LP5814I and LP5815. As for the LP5816 and LP5817, only the prior two pattern examples can be realized.

2 Hardware Setup

This section describes how to set up the connections between the LP5815YCHEVM evaluation module and the LP-MSPM0L1306 evaluation board. Items from the following list are required to begin the evaluation:

- Computer
- LP5815YCHEVM evaluation module
- LP-MSPM0L1306 evaluation board

Keep the default jumper settings on the LP-MSPM0L1306 evaluation board. The LP5815YCHEVM evaluation module can be powered directly from the 3.3V power of the LP-MSPM0L1306 evaluation board and does not need an external power supply. The power and I2C connections are demonstrated in Figure 2-1. The setup procedure is as follows:

1. Remove the jumper on the J3 header on the LP5815YCHEVM evaluation module. Keep the jumpers on the J2 and J9 headers to connect the voltage at the common collector (VCC) of the LP5815 device and the voltage for the LED (VLED) to the same 3.3V rail.
2. Refer to Figure 2-1 to connect the GND, SCL, SDA, STAT and 3.3V ports of the LP5815YCHEVM evaluation module to the LP-MSPM0L1306 evaluation board.
3. Connect the LP-MSPM0L1306 evaluation board to the computer through the USB cable carried with the kit.

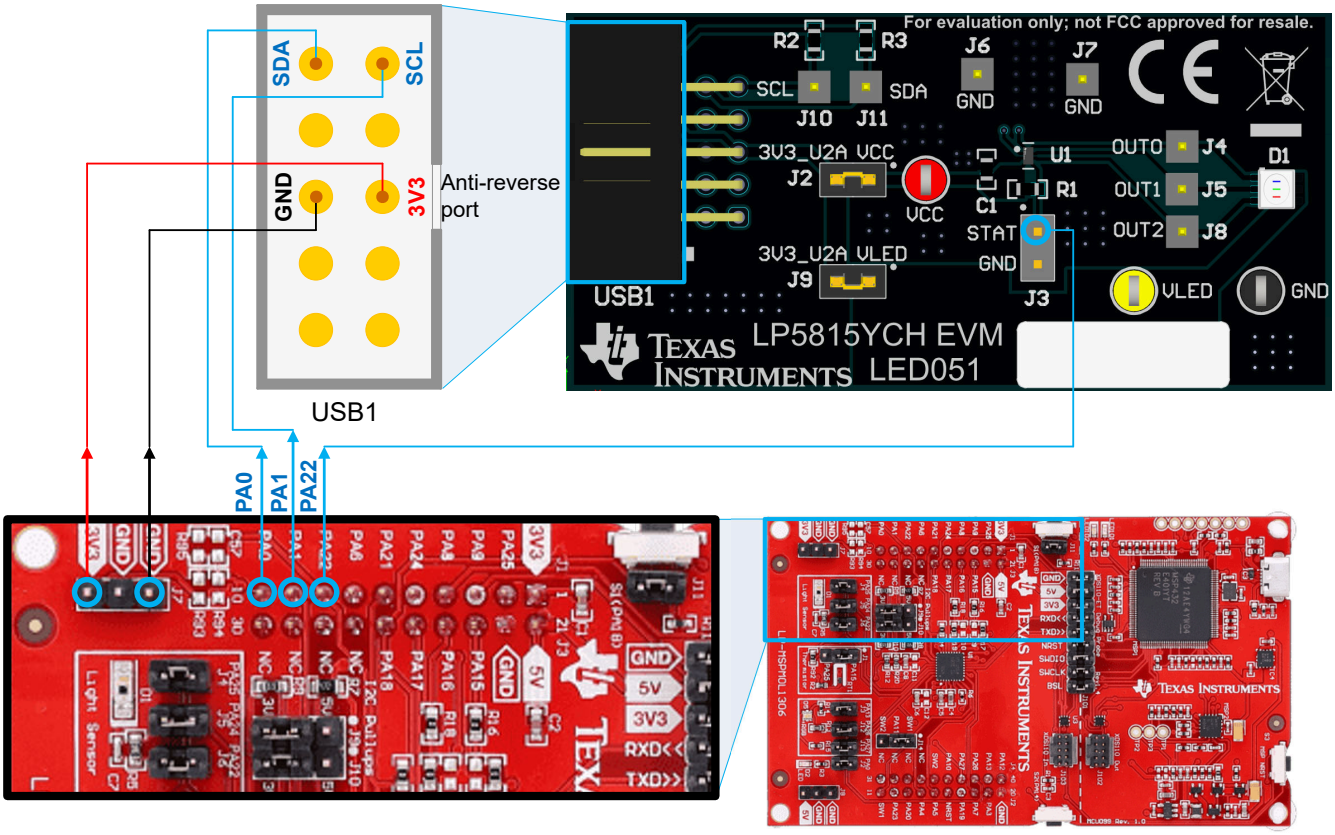

Figure 2-1. Hardware Connection

3 Software Setup

To set up the software for the LP-MSPM0L1306 LaunchPad™, please follow these steps (demonstrated in a computer with Windows 11 OS):

1. Download and install **Code Composer Studio™** .
   a. Download Code Composer Studio integrated development environment (IDE) (Version ≥ 12.5.0).
   b. Follow the installation instructions to install Code Composer Studio. During the installation process, if you choose the *Setup type* to be *Custom Installation*, select **MSPM0 32-bit Arm Cortex-M0+ General Purpose MCUs** in *Select Components*, as marked with the red box in Figure 3-1.

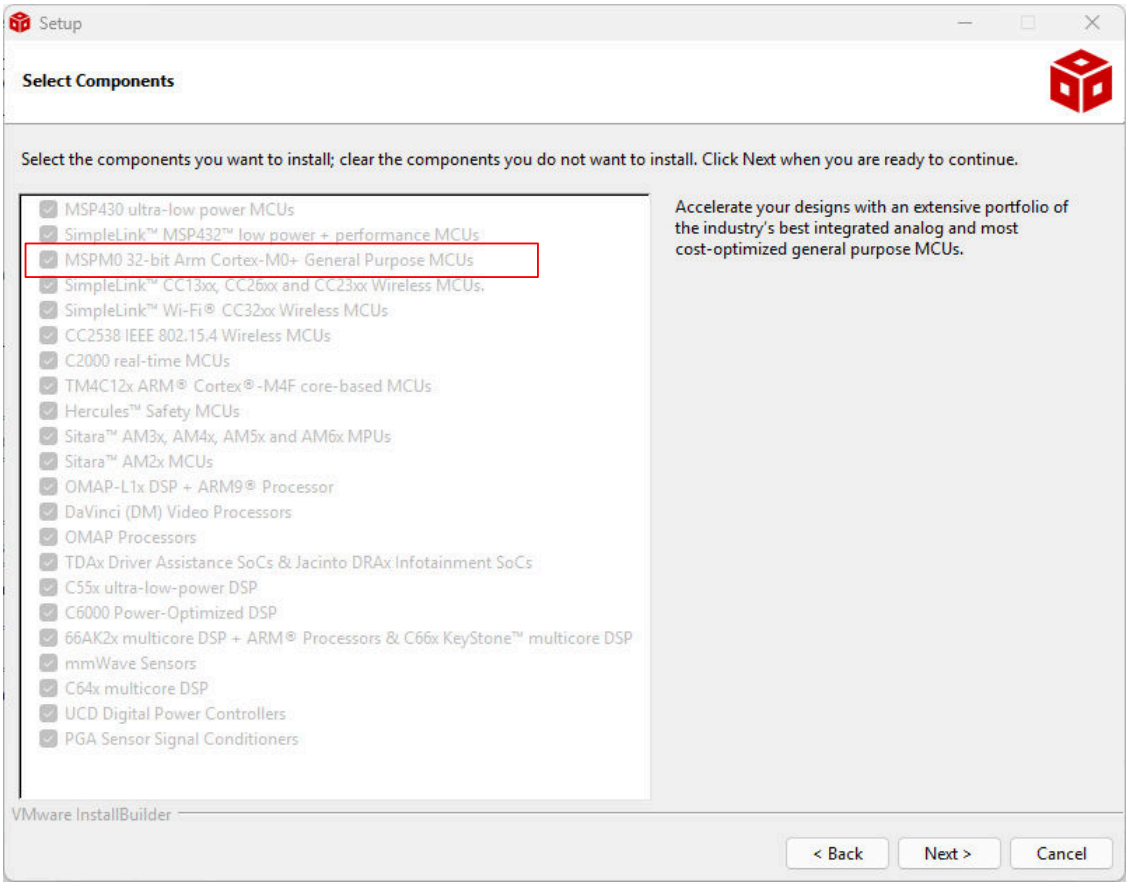

Figure 3-1. Installation Process of Code Composer Studio

2. Download and install MSPM0-SDK (Version ≥ 2.4.xx.xx).
   a. To verify the installation is correct, open the Code Composer Studio software. Select *Windows* from the top menu bar. Then select *Preferences* from the drop-down list. The *Preferences* window shows. From the left side bar, select *Code Composer Studio* → *Products*.
   b. Make sure that there is *MSPM0 SDK* and *SysConfig* (SysConfig is installed automatically when you install **MSPM0 32-bit Arm Cortex-M0+ General Purpose MCUs**) under the *Discovered products*, as is marked in Figure 3-2. If *MSPM0 SDK* or *SysConfig* does not appear, select *Refresh* on the right side of the *Preference* window. If *MSPM0 SDK* or *SysConfig* are still unavailable, check whether the installation is correct. A standalone desktop version of SysConfig can be found at SYSCONFIG System configuration tool.

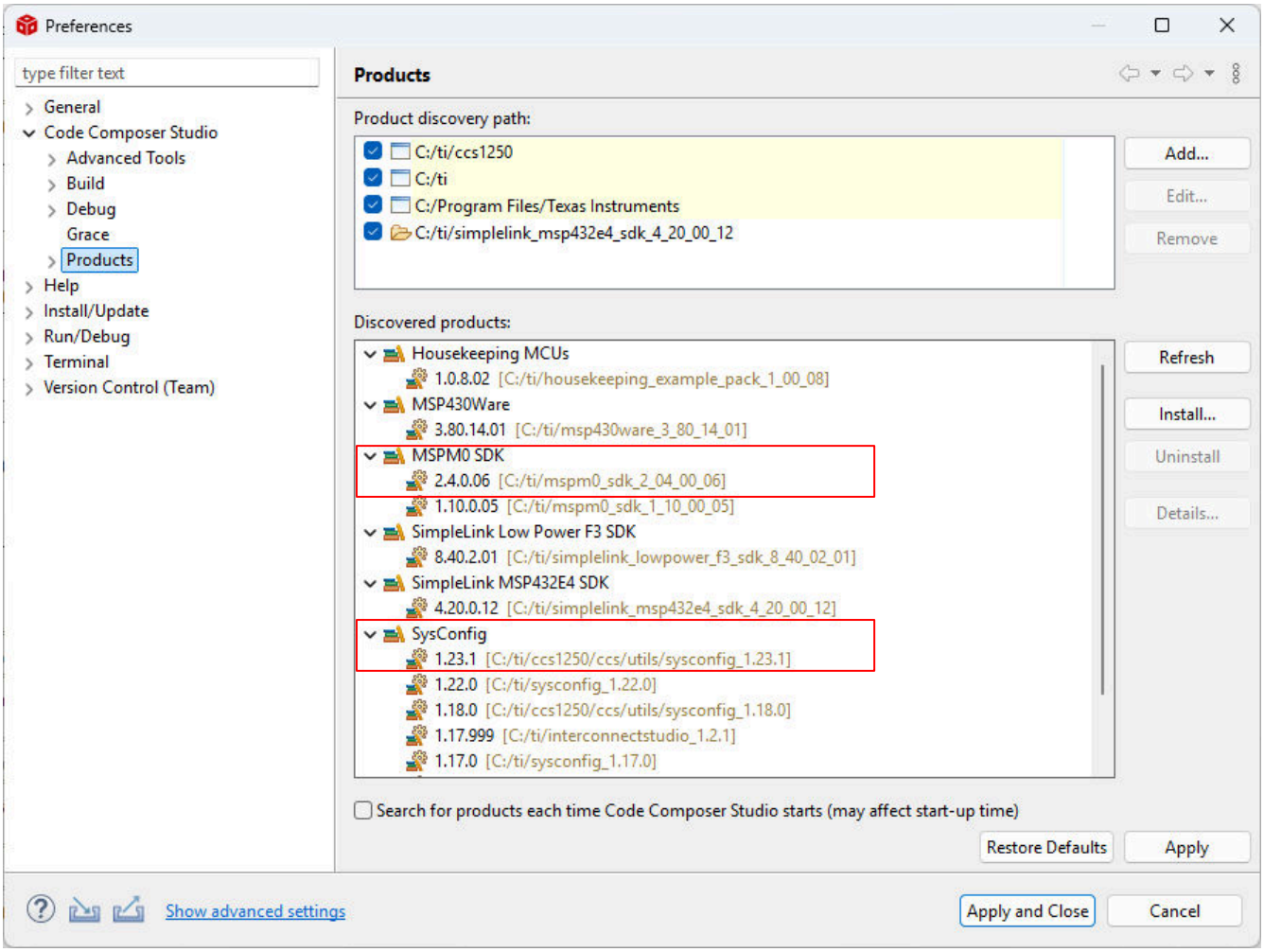

Figure 3-2. Verification of MSPM0 SDK Installation

3. Download and import the sample code. After downloading the sample code compressed file to a local computer, extract the file and import the Code Composer Studio (CCS) project according to the process provided in the link: Importing a CCS Project.
4. Load the program to the MCU according to the process provided in the link: Building and Running Your Project.

4 Sample Code Structure

4.1 Design Requirements

The design parameters for the LP5815 device are listed in Table 4-1.

Table 4-1. LP5815 Design Parameters

Design Parameter	LP5815
VCC and VLED	3.3V
OUT0 Max Current (Blue LED)	8mA
OUT1 Max Current (Green LED)	14mA
OUT2 Max Current (Red LED)	20mA
I2C Frequency	400kHz
I2C Target Address	0x2D

The I2C target address is defined as a macro in file lp5815_example.h. There are two address modes, independent and broadcast.

1. Independent address, *I2C_TARGET_ADDRESS_INDEPENDENT*, is different when communication occurs with different devices.
   a. 0x2D to communicate with LP5815 and LP5817.
   b. 0x2C to communicate with LP5814 and LP5816.
   c. 0x2E to communicate with LP5814I.
2. Broadcast address, *I2C_TARGET_ADDRESS_BROADCAST*, is the same when communication occurs with all devices. The address is 0x34.

```
#define I2C_TARGET_ADDRESS_INDEPENDENT (0x2D);  //b0010 1101 - LP5815, LP5817

#define I2C_TARGET_ADDRESS_BROADCAST (0x34);  //b0011 0100
```

4.2 Flow Diagram

Figure 4-1 depicts the high level flow in the sample code.

The three pattern examples are placed in the while loop. Refer to the Pattern Parameters section for a detailed description of the three pattern examples. The codes are placed in the file lp5815_example_patterns.h. The LP5815 device is configured in manual mode while executing the pattern example 1 and pattern example 2. The LP5815 device is configured in autonomous animation mode while executing the pattern example 3.

The step of sending *Update_command* is required to make the device configuration registers value take effect. **Send the *Update_command* after modifying the device configuration registers from *DEV_CONFIG0* to *DEV_CONFIG4*.**

```
/* send update cmd to make device configuration registers value take effect */
startRegAddress = UPDATE_CMD_REG;
tempData[0] = UPDATE_CMD_VAL; //data to write
i2cSendReceive(I2C_TARGET_ADDRESS_INDEPENDENT, startRegAddress, i2cWrite, 1, &tempData[0]);
```

When the *ENGINE_BUSY* flag in the *FLAG* register is set as 1, the engine configure registers and pattern configure registers shown in Table 4-2 are locked for modification protection. These engine busy lock registers can only be modified when **ENGINE_BUSY = 0**.

```
/*
 * need to check ENGINE_BUSY flag before,
 * engine configure registers, from ENGINE_CONFIG0 to ENGINE_CONFIGURE6,
 * and all pattern configure registers can be configured
 * read ENGINE_BUSY bit in FLAG register, the read back data stored in gRxPacket[]
 */
startRegAddress = FLAG_REG;
```

```
i2cSendReceive(I2C_TARGET_ADDRESS_INDEPENDENT, startRegAddress, i2cRead, 1, &tempData[0]);
/* send stop cmd to stop animation if ENGINE_BUSY = 1 */
if(gRxPacket[0] & 0x04)
{
    startRegAddress = STOP_CMD_REG;
    tempData[0] = STOP_CMD_VAL; //data to write
    i2cSendReceive(I2C_TARGET_ADDRESS_INDEPENDENT, startRegAddress, i2cWrite, 1, &tempData[0]);
}
```

Table 4-2. Engine Busy Lock Registers

Description	Register Address	Register Acronym
Engine configure registers	0x06 to 0x0C	ENGINE_CONFIG0 to ENGINE_CONFIG6
Pattern configure registers	0x1C to 0x3F	• PATTERN*x*_PAUSE_TIME • PATTERN*x*_REPEAT_TIME • PATTERN*x*_PWM0 • PATTERN*x*_PWM1 • PATTERN*x*_PWM2 • PATTERN*x*_PWM3 • PATTERN*x*_PWM4 • PATTERN*x*_SLOPER_TIME1 • PATTERN*x*_SLOPER_TIME2 *x* = 0, 1, 2, 3

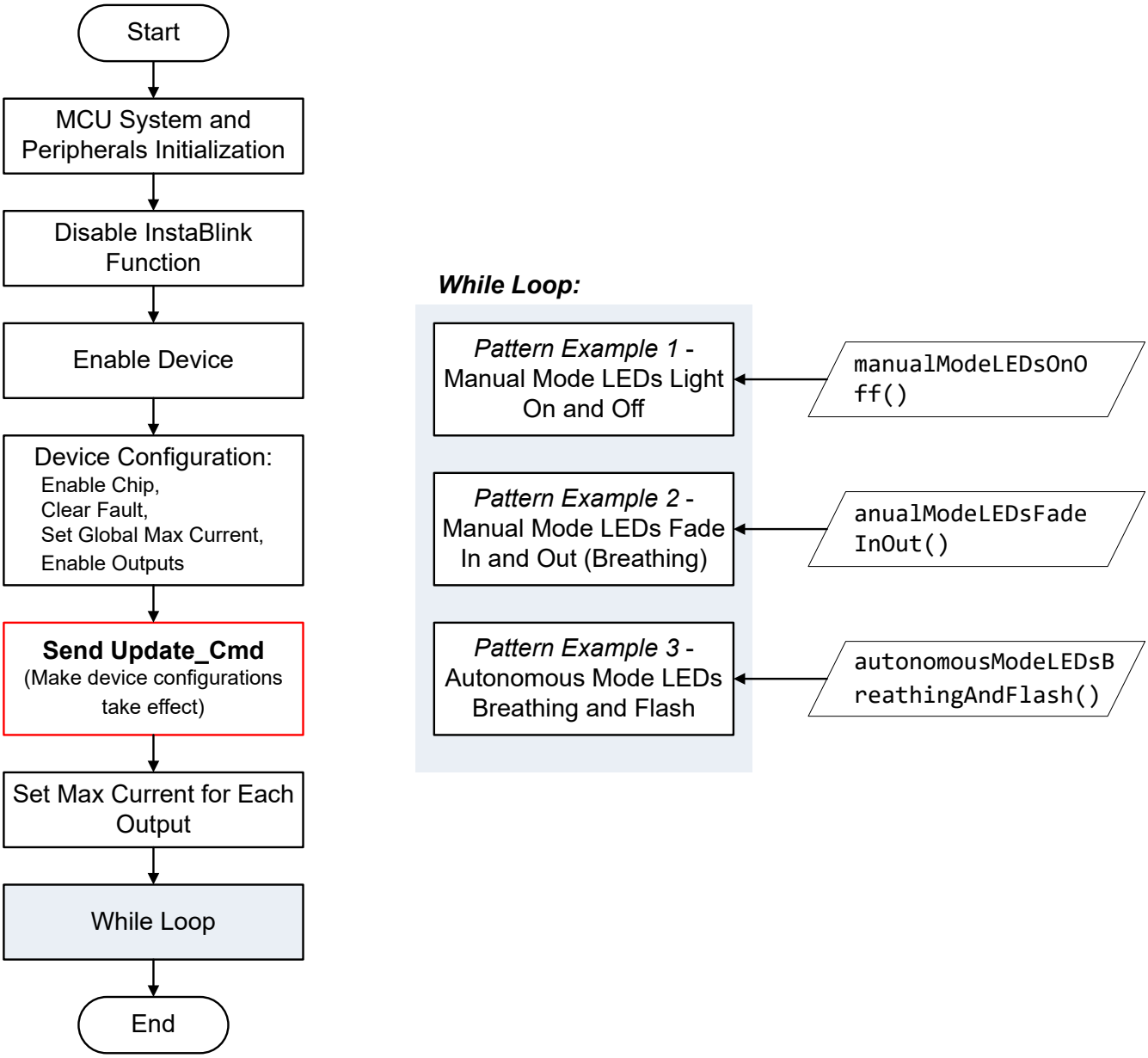

Figure 4-1. Sample Code Flow Diagram

4.3 Pattern Parameters

Pattern Example 1

The example corresponding function is manualModeLEDsOnOff(). In this example, the LP5815 is configured in manual mode. The red, green, and blue LEDs are controlled on and off simultaneously through writing 0x80 and 0x00 to the *OUTx_MANUAL_PWM (x = 0, 1, 2)* registers periodically. As shown in the Figure 4-2, the 500ms on and off duration is realized through the system delay function delay_ms().

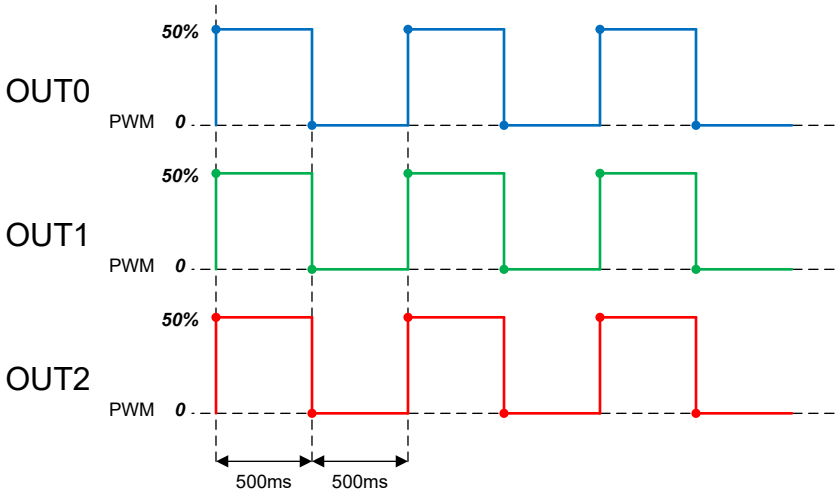

Figure 4-2. Pattern Example 1 Timing and Parameters

Pattern Example 2

The example corresponding function is manualModeLEDsFadeInOut(). In this example, the LP5815 is configured in manual mode.

To realize the fade in and out function, the sloper feature of the LP5815 is enabled through setting the *OUTx_FADE_EN (x = 0, 1, 2)* bits in the *DEV_CONFIG2* register. The outputs achieve 256 steps fade in or fade out effects from current PWM value to the updated PWM value autonomously to save MCU communication loading. To obtain a better human-eye-friendly dimming effect, the exponential scale is enabled through setting the *OUTx_EXP_EN (x = 0, 1, 2)* bits in the *DEV_CONFIG3* register. As shown in Figure 4-3, the 1s fade in and out time is realized through setting the *LED_FADE_TIME* bits as 0xB in the *DEV_CONFIG2* register.

Make sure *Update_command* is sent to make the device configuration registers value take effect.

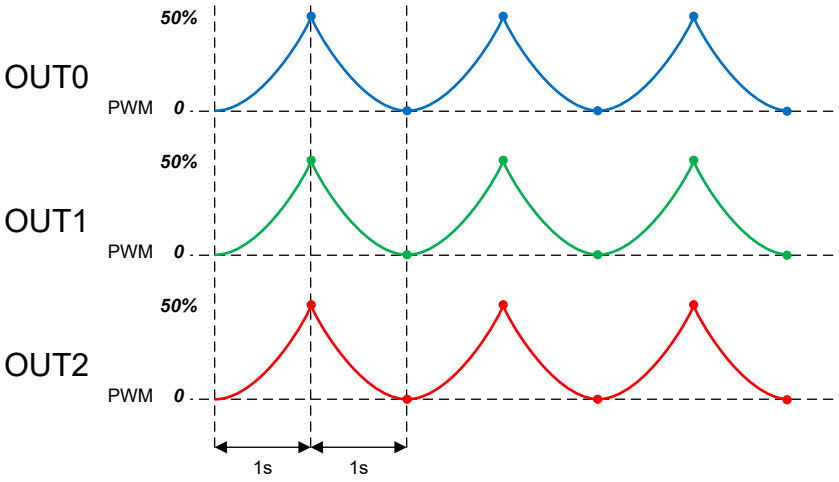

Figure 4-3. Pattern Example 2 Timing and Parameters

Pattern Example 3

The example corresponding function is autonomousModeLEDsBreathingAndFlash(). In this example, the LP5815 is configured in autonomous animation mode through setting the *OUTx_AUTO_EN (x = 0, 1, 2)* bits in the *DEV_CONFIG3* register.

The OUT0 selects the ENGINE0, OUT1 selects the ENGINE1, and OUT2 selects ENGINE2 through setting the *OUTx_ENGINE_CH (x = 0, 1, 2)* bits in the *DEV_CONFIG4* register. Corresponding codes are shown below.

Make sure *Update_command* is sent to make device configuration registers value take effect.

```
/*
 * OUT2 select ENGINE 2
 * OUT1 select ENGINE 1
 * OUT0 select ENGINE 0
 */
startRegAddress = DEV_CONFIG4;
tempData[0] = 0x24; //data to write
i2cSendReceive(I2C_TARGET_ADDRESS_INDEPENDENT, startRegAddress, i2cWrite, 1, &tempData[0]);
```

Check the *ENGINE_BUSY* flag is 0 before setting the engine configuration registers from *ENGINE_CONFIG0* to *ENGINE_CONFIG6*.

Enable the engine order and select patterns for the engine orders to construct the patterns for OUT0, OUT1, and OUT2. Two engine orders are enabled for each engine through setting the *ExOy_EN (x = 0, 1, 2; y = 0, 1)* bits in the *ENGINE_CONFGIG4* and *ENGINE_CONFGIG5* registers. Corresponding codes are shown below.

```
/*
 * enable ENGINE0_ORDER0 and ENGINE0_ORDER1,
 * enable ENGINE1_ORDER0 and ENGINE1_ORDER1,
 * enable ENGINE2_ORDER0 and ENGINE2_ORDER1
 */
startRegAddress = ENGINE_CONFIG4;
tempData[0] = 0x33; //data to write
tempData[1] = 0x03;
i2cSendReceive(I2C_TARGET_ADDRESS_INDEPENDENT, startRegAddress, i2cWrite, 2, &tempData[0]);
```

Select the patterns for the engine orders to construct the final patterns for OUT0, OUT1, and OUT2 through setting the *ENGINEx_ORDERy (x = 0, 1, 2; y = 0, 1)* bits in the *ENGINE_CONFGIG0* to *ENGINE_CONFGIG2* registers. Corresponding codes are shown below.

```
/*
 * ENGINE0: ENGINE0_ORDER0 select PATTERN0, ENGINE0_ORDER1 select PATTERN3, (0x0C)
 * ENGINE1: ENGINE1_ORDER0 select PATTERN1, ENGINE1_ORDER1 select PATTERN3, (0x0D)
 * ENGINE2: ENGINE2_ORDER0 select PATTERN2, ENGINE2_ORDER1 select PATTERN3, (0x0E)
 */
startRegAddress = ENGINE_CONFIG0;
tempData[0] = 0x0C; //data to write
tempData[1] = 0x0D;
tempData[2] = 0x0E;
i2cSendReceive(I2C_TARGET_ADDRESS_INDEPENDENT, startRegAddress, i2cWrite, 3, &tempData[0]);
```

The entire engine pattern repeats two times through setting the *ENGINEx_REPT (x = 0, 1, 2)* bits in the *ENGINE_CONFGIG6* register.

The LP5815 has four independent configurable pattern units, *PATTERN0* to *PATTERN3*. In this example, PATTERN0, PATTERN1, PATTERN2, and PATTERN3 are all used. Refer to the Figure 4-4 for the pattern parameters setting for each pattern unit. The function of setPatternParameters() in the lp5815_example_patterns.h file is used to configure all the parameters for each pattern unit as shown below.

Check the *ENGINE_BUSY* flag is 0 before setting the pattern parameters configuration registers from *PATTERN0_PAUSE_TIME* to *PATTERN3_SLOPER_TIME2*.

```
/*
 *  @brief      set pattern parameters
 *
 *  @param[in]  patternBaseAdr  i2c slave address
 *  @param[in]  pauseT0  start register address to write or read
 *  @param[in]  pauseT1  write or read operation
 *  @param[in]  pt data length of the data to write or read
 *  @param[in]  pwm0/pwm1/pwm2/pwm3/pwm4  point to the data array to write
 *  @param[in]  sloperT0/sloperT1/sloperT2/sloperT3
 */
void setPatternParameters(uint8_t patternBaseAdr, uint8_t pauseT0, uint8_t pauseT1, uint8_t pt,
                          uint8_t pwm0, uint8_t pwm1, uint8_t pwm2, uint8_t pwm3, uint8_t pwm4,
                          uint8_t sloperT0, uint8_t sloperT1, uint8_t sloperT2, uint8_t sloperT3)
{
    uint8_t startRegAddress; //the start address to write or read from
```

```
    startRegAddress = patternBaseAdr;
    tempData[0] = (pauseT0 << 4) | (pauseT1); //data to write
    tempData[1] = pt;
    tempData[2] = pwm0;
    tempData[3] = pwm1;
    tempData[4] = pwm2;
    tempData[5] = pwm3;
    tempData[6] = pwm4;
    tempData[7] = (sloperT1 << 4) | (sloperT0);
    tempData[8] = (sloperT3 << 4) | (sloperT2);
    i2cSendReceive(I2C_TARGET_ADDRESS_INDEPENDENT, startRegAddress, i2cWrite, 9, &tempData[0]);
}
```

After completing all pattern parameters setting, **send *Start_command* to start the autonomous animation running on the outputs**. The *ENGINE_BUSY* flag is set as 1 to indicate there is autonomous patterns running on the output. When the patterns stopped, the *ENGINE_BUSY* flag is cleared automatically.

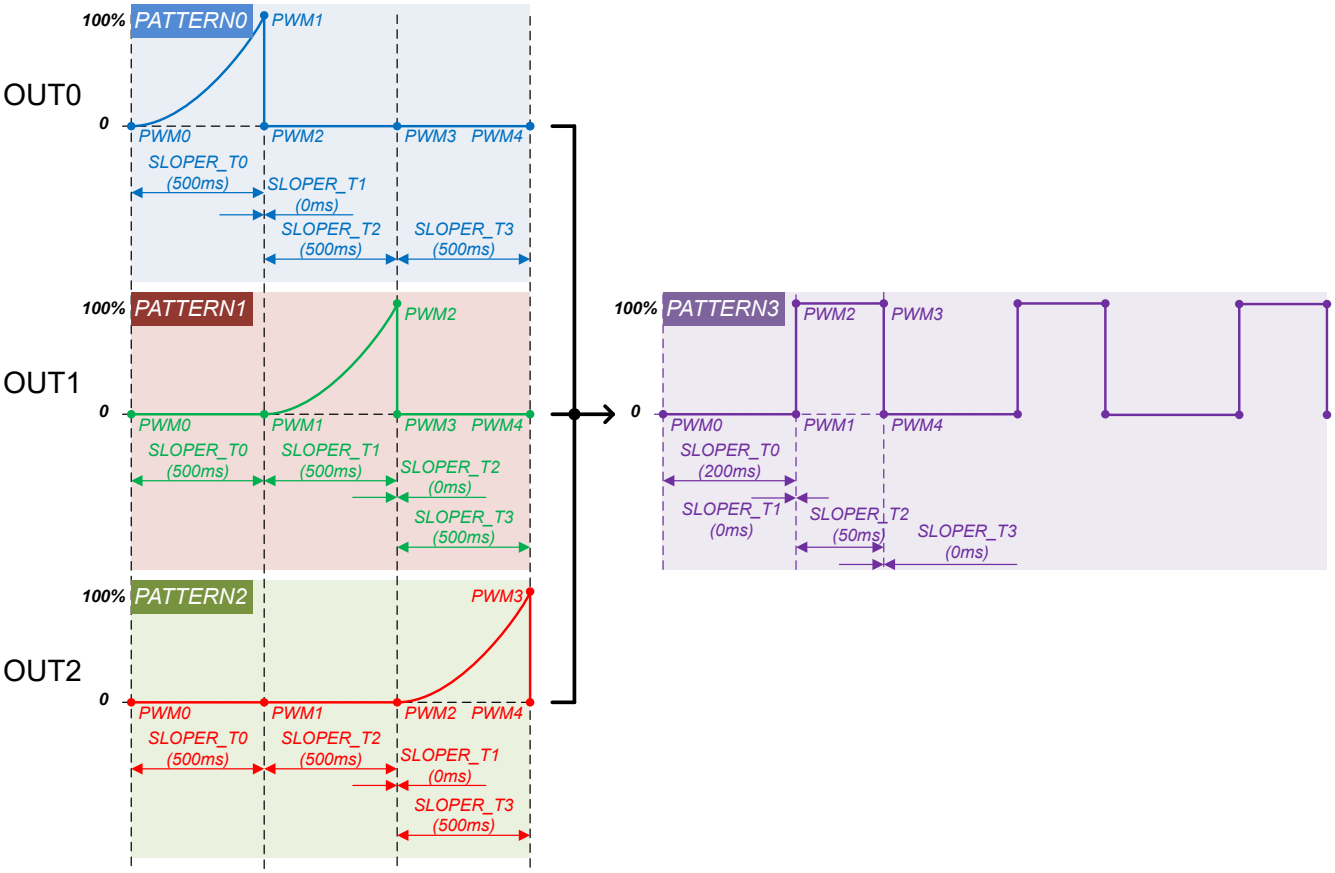

Figure 4-4. Pattern Example 3 Timing and Parameters

5 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

DATE	REVISION	NOTES
July 2026	*	Initial Release

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025