Application Note

TPS929xxx LED Driver Control Using MSPM0 Through UART Over CAN

Helic Chi

ABSTRACT

This application note is a guide for the preparation and usage of the sample code for the TPS929xxx-Q1 device family paired with a LP-MSPM0G3507. Users can also port this code easily to other MSPM0 devices using the system configuration tool (SYSCONFIG).

Table of Contents

Trademarks

1 Introduction

The sample code showcases the ability to light up the LEDs on the TPS929120EVM, TPS929160EVM, and TPS929240EVM. Each TPS929xxxEVM has own sample code. However, the only difference is in the file led_driver.h to select the used LED driver IC. This helps the user to be able to light up the EVM without any modification to the sample code.

There are two modes in the code: animation and EEPROM programming. The animation mode is selected by default. How to change between the modes is described in Section 4.2. In the animation mode, 6 different animations are used according to a predefined sequence. Switching between the different animations is achieved by clicking on button S2 on the LP-MSPM0G3507. After every animation, diagnostics is executed to determine if any fault occurred. For more information about diagnostics, refer to Section 4.3.

The sample code comes with many predefined APIs that can be used to change the configuration of the LED driver, perform diagnostics, or build custom FlexWire commands. The predefined APIs automatically adjust to the specified system. More detail about the system specification can be found in Section 4.2.

2 Hardware Setup

This section describes the differences in the hardware setup compared to the description in each of the device-specific EVM User's Guides. Check the hardware setup in:

- Tool pages:
  - TPS929120EVM
  - TPS929160EVM
  - TPS929240EVM
- Documentation:
  - TPS929120EVM User's Guide
  - TPS929160EVM User's Guide
  - TPS929240EVM User's Guide

The sample code replaces the USB2ANY with the LP-MSPM0G3507. There are two methods to connect the LP-MSPM0G3507 to the TPS929xxxEVM, the TPS929240EVM is used for demonstration here.

Both connection methods are listed in Table 2-1. The locations on the LP-MSPM0G3507 are shown in Figure 2-1. Firstly, jump the SW1 and PA9 of J14 together. Besides the connections, the switches S1 (cyan) and S2 (green) of the LP-MSPM0G3507are used in the sample code that is described in more detail in Section 4.1. On the TPS929xxxEVM, the jumpers have to be set correctly for each mode. This is described in the device-specific EVM User's Guides.

Table 2-1. Hardware Connections

Interface	Board	UART-RX	UART-TX	+3.3V	GND	+5V
	LP-MSPM0G3507	PA9	PA8	J1-1	J1-22	J1-21
CAN	TPS929120CANEVM	J3-13	J3-14	J3-15	J3-16	+5V (J3-21)
UART	TPS929120EVM	J3-3	J3-4	J3-5	J3-6	\
	TPS929160EVM	J29-3	J29-4	J29-5	J29-6	\
	TPS929240EVM	J4-3	J4-4	J4-5	J4-6	\

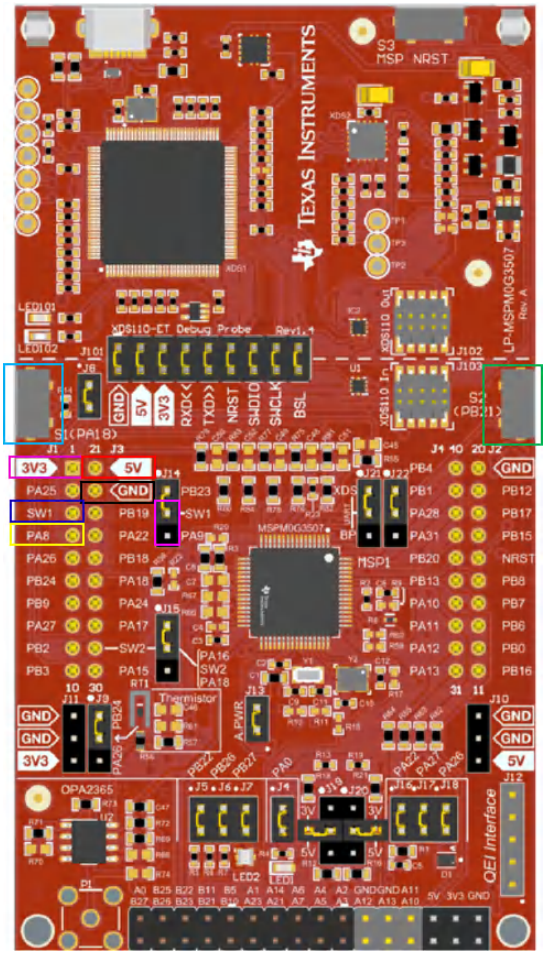

Figure 2-1. Connections on LP-MSPM0G3507

3 Software Setup

To set up the software for the LP-MSPM0G3507, follow these steps (demonstrated in a computer with Windows® 10 OS):

1. Download and install Code Composer Studio™ (CCS): Code Composer Studio integrated development environment (IDE). Follow online guide to install CCS or use the MSPM0 Design Flow Guide.
   a. In the installation, when CCS installer is in *Select Components* page, select *MSPM0 32-bit Arm Cortex-M0+ General Purpose MCUs*.
2. Download MSPM0-SDK and import sample code.
   a. Download MSPM0-SDK from ti.com.
   b. In CCS, click Project - Import CCS Projects, choose the TPS929xxx demo SDK folder path: {SDK folder}\examples\nortos\LP_MSPM0G3507\demos\TPS929xxx_control_uart_over_can, import demo code.
3. Refer to the TPS929120EVM, TPS929160EVM or TPS929240EVM's user's guides to change the EVM address. Synchronously change the device number *DEVICE_CNT* and device address *device_address* in the system_info.h and system_info.c file.
4. Build and load the program.
5. Press S2 button on LP-MSPM0G3507, TPS929xxxEVM changes the LED pattern.
6. (Optional) Download the EEPROM configuration tool. This is a good tool if users want to program the EEPROM with non-default data. The tool for TPS929160/TPS929240 also includes a calculation tool for the FlexWire interface CRC values in tab IF_CRC. For each of the supported devices, there is a separate link:
   a. TPS929120/TPS929121: TPS92912x-Q1 EEPROM Configuration Tool.
   b. TPS929160/TPS929240: TPS929240-Q1 TPS929160-Q1 EEPROM Configuration Tool.

4 Software Structure

4.1 Flow Diagram

The high-level flow in the sample code is shown in Figure 4-1. Throughout the flow, the number of devices and addresses on the FlexWire bus are used. This is specified in the files system_info.h and system_info.c and is described in more detail in Section 4.2.

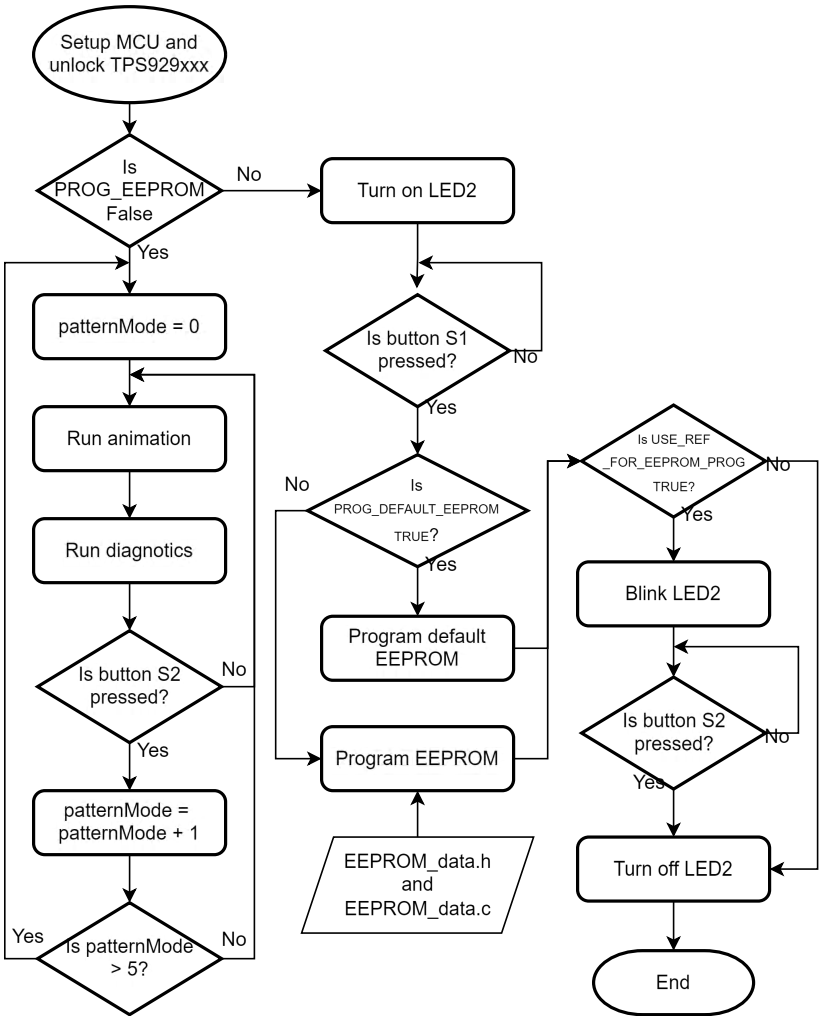

Figure 4-1. Software Flow Diagram

The Setup MCU configures the UART interface and sets to 750000 bauds. After unlocking the LED driver, M0 check if the animation mode or EEPROM programming mode has been selected. How to change between the modes is described in Section 4.2. During the animation mode, a LED pattern is executed and after completion the diagnostics results are checked. More information about diagnostics can be found in Section 4.3. After the diagnostics, M0 check if the button S2 on the LP-MSPM0G3507 was pressed. If button is not pressed, the same LED pattern executes again. If button is pressed, the next LED pattern executes until all 6 patterns have been executed and the loop restart from the first pattern again.

During the EEPROM programming mode, both buttons S1 and S2 on LP-MSPM0G3507 are used and LED2 to provide feedback to the user. When the non-default EEPROM programming is selected, files eeprom_data.h and eeprom_data.c are used to program the EEPROM. These files can be automatically generated by the EEPROM Configuration Tool mentioned in Section 2. More information about EEPROM programming can be found in Section 4.4.

4.2 System Setup

This section describes how the sample code sets different parameters to identify how the system is built. The first part is the actual used LED driver IC. Within the led_driver.h file, the used LED driver IC is selected. The example code includes TPS929240 in default.

The example code also supports:

- TPS929120
- TPS929120A
- TPS929121
- TPS929121A
- TPS929160
- TPS929240
- TPS929240A

Note that, for the *Q1* devices, this is not added and only the base product name is used. The selected device is important to handle different register addresses and fields in registers. Moreover, when the default EEPROM is being programmed, the specified LED driver IC is the one being used to program the default value. This means that when the user wants to program a TPS929120 to TPS929120A, TPS929120A has to be selected in the file led_driver.h.

A summary of macros and variables that impact the system setup is listed in Table 4-1.

Table 4-1. Summary of Macro and Variable Names per File

Filename	Macro or Variable Name	Description
system_info.h	DEVICE_CNT	Number of devices on the FlexWire bus
	CAN_USED	Selection between UART or UART-over-CAN
	ALWAYS_CHECK_CRC	Enable CRC check for all non-broadcast commands
	PROG_EEPROM	Enable EEPROM programming mode
	PROG_DEFAULT_EEPROM	Program default EEPROM values instead of custom defined EEPROM values
	USE_REF_PIN_FOR_EEPROM_PROG	Use REF-pin during EEPROM programming
system_info.c	device_address	List of device addresses on the FlexWire bus
FlexWire.c	rcvCrcError	Report if received CRC has error

Within the file system_info.h the number of devices on the FlexWire bus is defined by macro DEVICE_CNT. The sample code only support 1 FlexWire bus.

```
// Total devices on FlexWire bus
#define DEVICE_CNT 1
```

The actual addresses of the devices are specified in file system_info.c. The sequence of addresses determines the order of FlexWire non-broadcast write and read commands. Therefore, the LED patterns in the animation mode look different for different sequences of device addresses.

```
const uint16_t device_address[DEVICE_CNT] = {DEVICE_ADDR__1};
```

File system_info.h also defines other system parameters.

```
// Define if CAN or UART is used
#define CAN_USED FALSE
// When non-broadcast is transmitted, does the CRC need to be checked
#define ALWAYS_CHECK_CRC FALSE
```

Macro *CAN_USED* defines if UART or UART-over-CAN is being used for the FlexWire bus. This impacts the total number of bytes that are being received on the MCU UART-RX pin.

Macro *ALWAYS_CHECK_CRC* defines if for the received feedback, the CRC needs to be checked for every non-broadcast write command. When the CRC is checked and found incorrect, global variable *rcvCrcError* is set to TRUE. In all other cases, this variable is set to FALSE. The variable *rcvCrcError* is defined in file FlexWire.c.

```
// When an error in CRC of the received data is observed, set this to TRUE
unsigned int rcvCrcError;
```

4.3 Diagnostics

The sample code provides an API to detect which devices have faults such as open, short, or single-LED-short. Within the TPS929xxx_APIs.h file the prototype of the API is defined.

```
void LED_Update_Chip_Status(unsigned int dev_addr_x);
```

This API updates the variable *chip_status* which is defined in system_info.h. For devices TPS929160-Q1 and TPS929240-Q1, there is an additional power pin called VBAT. Therefore, the variable includes a measured voltage result for this pin for those devices. In addition, these devices include an additional fault type called Supply Undervoltage. Therefore, these devices include flag SUPUV.

```
struct chipStatus {
// Indicates open, short, and/or single-LED-short fault
uint16_t OUT_flag;
uint16_t SHORT_channels[MAX_CHANNEL_CNT];
uint16_t OPEN_channels[MAX_CHANNEL_CNT];
uint16_t SLS_channels[MAX_CHANNEL_CNT]; // Single-LED-short
uint16_t EEPCRC; // EEPROM CRC fault
uint16_t TSD; // Thermal Shutdown
uint16_t PRETSD; // Pre-thermal shutdown warning
uint16_t REF; // REF-pin fault
uint16_t LOWSUP; // Low supply
uint16_t POR; // Power-on-reset
#ifndef TPS92912X
uint16_t SUPUV; // Supply undervoltage
uint16_t VBAT_mV; // *1 mV
#endif
uint16_t VSUPPLY_mV; // *1 mV
uint16_t VLDO_mV; // *1 mV
uint16_t TEMPSNS_10mC; // *10 mC
uint16_t VREF_100uV; // *100 uV
uint16_t IREF_10nA; // *10 nA
};
// For diagnostics
extern struct chipStatus chip_status[];
```

The variable *chip_status* can be watched in the Expressions view during the debug of the code by following the steps in Watching Variables, Expressions, and Registers. An example without any error is depicted in Figure 4-2. The first index of variable *chip_status* is the address of the LED driver on the FlexWire bus. In total there are 16 different addresses. Therefore, the index runs from 0 to 15.

An example with a short is depicted in Figure 4-3. The TPS929240-Q1 has address 0x1 and flag OUT_Flag is set. When the array SHORT_channels is expanded, the short occurs on pin OUT2.

Memory Browser | (x)= Variables | Expressions | Registers

Expression	Type	Value
chip_status	struct chipStatus[16]	[{OUT_flag=0,SHORT_channels=[0,0,0,0,0.....
[0]	struct chipStatus	{OUT_flag=0,SHORT_channels=[0,0,0,0,0...]...
[1]	struct chipStatus	{OUT_flag=0,SHORT_channels=[0,0,0,0,0...]...
OUT_flag	unsigned short	0
SHORT_channels	unsigned short[24]	[0,0,0,0,0...]
OPEN_channels	unsigned short[24]	[0,0,0,0,0...]
SLS_channels	unsigned short[24]	[0,0,0,0,0...]
EEPCRC	unsigned short	0
TSD	unsigned short	0
PRETSD	unsigned short	0
REF	unsigned short	0
LOWSUP	unsigned short	0
POR	unsigned short	0
SUPUV	unsigned short	0
VBAT_mV	unsigned short	0
VSUPPLY_mV	unsigned short	0
VLDO_mV	unsigned short	0
TEMPSNS_10mC	unsigned short	0
VREF_100uV	unsigned short	0
IREF_10nA	unsigned short	0
[2]	struct chipStatus	{OUT_flag=0,SHORT_channels=[0,0,0,0,0...]...

Figure 4-2. Example of chip_status Without Errors for TPS929240-Q1

Memory Browser | (x)= Variables | Expressions | Registers

Expression	Type	Value
chip_status	struct chipStatus[16]	[{OUT_flag=0,SHORT_channels=[0,0,0,0,0.....
[0]	struct chipStatus	{OUT_flag=0,SHORT_channels=[0,0,0,0,0...]...
[1]	struct chipStatus	{OUT_flag=1,SHORT_channels=[0,0,1,0,0...]...
OUT_flag	unsigned short	1
SHORT_channels	unsigned short[24]	[0,0,1,0,0...]
[0]	unsigned short	0
[1]	unsigned short	0
[2]	unsigned short	1
[3]	unsigned short	0

Figure 4-3. Example of chip_status With Errors for TPS929240-Q1

An example of when a low supply warning occurs (V(SUPPLY) < V(ADCLOWSUPTH)) in TPS929240-Q1 is depicted in Figure 4-4. The flag LOWSUP has been set for a device with address 0x1. In addition, for this warning, the supply voltage is measured by the ADC and reported in the diagnostics as well. The measured result in this example is 4804mV.

Memory Browser | (x)= Variables | Expressions | Registers

Expression	Type	Value
chip_status	struct chipStatus[16]	[{OUT_flag=0,SHORT_channels=[0,0,0,0,0.....
[0]	struct chipStatus	{OUT_flag=0,SHORT_channels=[0,0,0,0,0...]...
[1]	struct chipStatus	{OUT_flag=0,SHORT_channels=[0,0,0,0,0...]...
OUT_flag	unsigned short	0
SHORT_channels	unsigned short[24]	[0,0,0,0,0...]
OPEN_channels	unsigned short[24]	[0,0,0,0,0...]
SLS_channels	unsigned short[24]	[0,0,0,0,0...]
EEPCRC	unsigned short	0
TSD	unsigned short	0
PRETSD	unsigned short	0
REF	unsigned short	0
LOWSUP	unsigned short	1
POR	unsigned short	0
SUPUV	unsigned short	0
VBAT_mV	unsigned short	0
VSUPPLY_mV	unsigned short	4804
VLDO_mV	unsigned short	0
TEMPSNS_10mC	unsigned short	0
VREF_100uV	unsigned short	0
IREF_10nA	unsigned short	0
[2]	struct chipStatus	{OUT_flag=0,SHORT_channels=[0,0,0,0,0...]...

Figure 4-4. Example of chip_status With Low Supply for TPS929240-Q1

4.4 EEPROM Programming

The sample code includes the capability to program the EEPROM. This capability is enabled by macros defined in file system_info.h.

```
// When set to 1, the EEPROM programming routine is executed instead of normal program
#define PROG_EEPROM             (FALSE)
// When set to 1, program the EEPROM to the default value
#define PROG_DEFAULT_EEPROM         (FALSE)
// Use external device address settings for EEPROM programming
#define USE_REF_PIN_FOR_EEPROM_PROG     (FALSE)
```

When macro *PROG_EEPROM* is defined as TRUE, the *EEPROM* programming mode is enabled. The sample code can program the default EEPROM values for the specified LED driver IC or a custom setting. The custom setting is programmed when macro *PROG_DEFAULT_EEPROM* is defined as FALSE. This setting is specified in files eeprom_data.h and eeprom_data.c. These files can be automatically generated by the EEPROM Configuration Tool as mentioned in Section 3.

The LED driver IC supports two method for individual chip selection through pulling the REF pin high or through device address configuration by address pin. When macro *USE_REF_PIN_FOR_EEPROM_PROG* is defined as TRUE, the REF pin is pulled high during programming. When *USE_REF_PIN_FOR_EEPROM_PROG* is defined as FALSE, the current device address is used. TI recommends using the current device address.

When the code enters the EEPROM programming routine, M0 turns on LED2 (PB27) on the LP-MSPM0G3507. When macro *USE_REF_PIN_FOR_EEPROM_PROG* is defined as TRUE, the REF pin is pulled up after the LED2 turns on. The jumper required to pull up the REF-pin for each EVM is listed in Table 4-2.

After LED2 turns on, button S1 on LP-MSPM0G3507 need to be pressed to start the programming. When the current device address is used, LED2 turns off once the programming is finished.

When the REF pin is used, LED2 starts to blink after the programming is finished. At that time, the pull up on the REF pin is removed and afterward the button S2 on LP-MSPM0G3507 need to be pressed. Then LED2 turns off.

Table 4-2. EVM Jumper to be Set When Using REF PIN During EEPROM Programming

EVM	Jumper
TPS929120EVM	J2 position 2 and 3 (+5V)
TPS929160EVM	J52 position 2 and 3 (VLDO)
TPS929240EVM	J10 position 2 and 3 (VLDO)

5 Summary

Automotive lighting is widely used in automobile headlights, taillights, and ambient lighting. With the increasing demand for animation in automotive lighting, LEDs must be controlled independently. Therefore, there is a need for better MCU and designs to meet new requirement. The MSPM0G-series with TPS929xxxEVM gives designs of independently LEDs control.

This application note introduces the hardware connection between LP-MSPM0G3507 and TPS929xxxEVM and how to setup the test and debug environment from zero, including IDE and SDK installation and code import steps.

The application note also supplies the software diagram, software system parameter setup and basic function of demo code and shows how to test the chip error status through simple steps and gives the results for comparison.

6 References

1. Texas Instruments, TPS929xxx-Q1 Sample Code, user's guide.
2. Texas Instruments, TPS929240-Q1 High-Side (O)LED Driver With FlexWire Interface, data sheet.
3. Texas Instruments, TPS929160-Q1 High-Side (O)LED Driver With FlexWire Interface, data sheet.
4. Texas Instruments, TPS929120-Q1 High-Side (O)LED Driver With FlexWire Interface, data sheet.
5. Texas Instruments, TPS929240EVM User's Guide, user's guide.
6. Texas Instruments, TPS929160EVM User's Guide, user's guide.
7. Texas Instruments, TPS929120EVM User's Guide, user's guide.
8. Texas Instruments, MSPM0G350x Mixed-Signal Microcontrollers With CAN-FD Interface, data sheet.
9. Texas Instruments, MSPM0 G-Series 80MHz Microcontrollers Technical Reference Manual, user's guide.
10. Texas Instruments, MSPM0G3507 LaunchPad Development Kit User's Guide (LP-MSPM0G3507), EVM user's guide.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), 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 will 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 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.

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

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265