User's Guide

TPS929xxx-Q1 Sample Code

ABSTRACT

This document acts as a guide for the preparation and usage of the sample code for the TPS929xxx-Q1 device family paired with a MSP-EXP430F5529LP. The code that can be downloaded from the website will be able to light up the LEDs for each matching EVM.

Table of Contents

List of Figures

List of Tables

Trademarks

1 Introduction

The sample code showcases the ability to light up the LEDs on the TPS929120EVM, TPS929160EVM, and TPS929240EVM. Each EVM has its 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. Section 4.2 describes how to change between the modes. 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 MSP-EXP430F5529LP. After every animation, diagnostics is executed to determine if any fault occurred. For more information about diagnostics see 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 Software Setup

To set up the software for the MSP430F5529 LaunchPad™, follow these steps (demonstrated in a computer with Windows 10 OS):

1. Download and install Code Composer Studio™
   a. Download Code Composer Studio integrated development environment (IDE) (Version ≥ 11.1.0).
   b. Follow the installation instructions to install Code Composer Studio. During the installation process, if you choose the "Setup type" to be "Custom Installation", make sure that you select "MSP430 ultra-low power MCUs" in "Select Components", as is marked with red box in Figure 2-1.

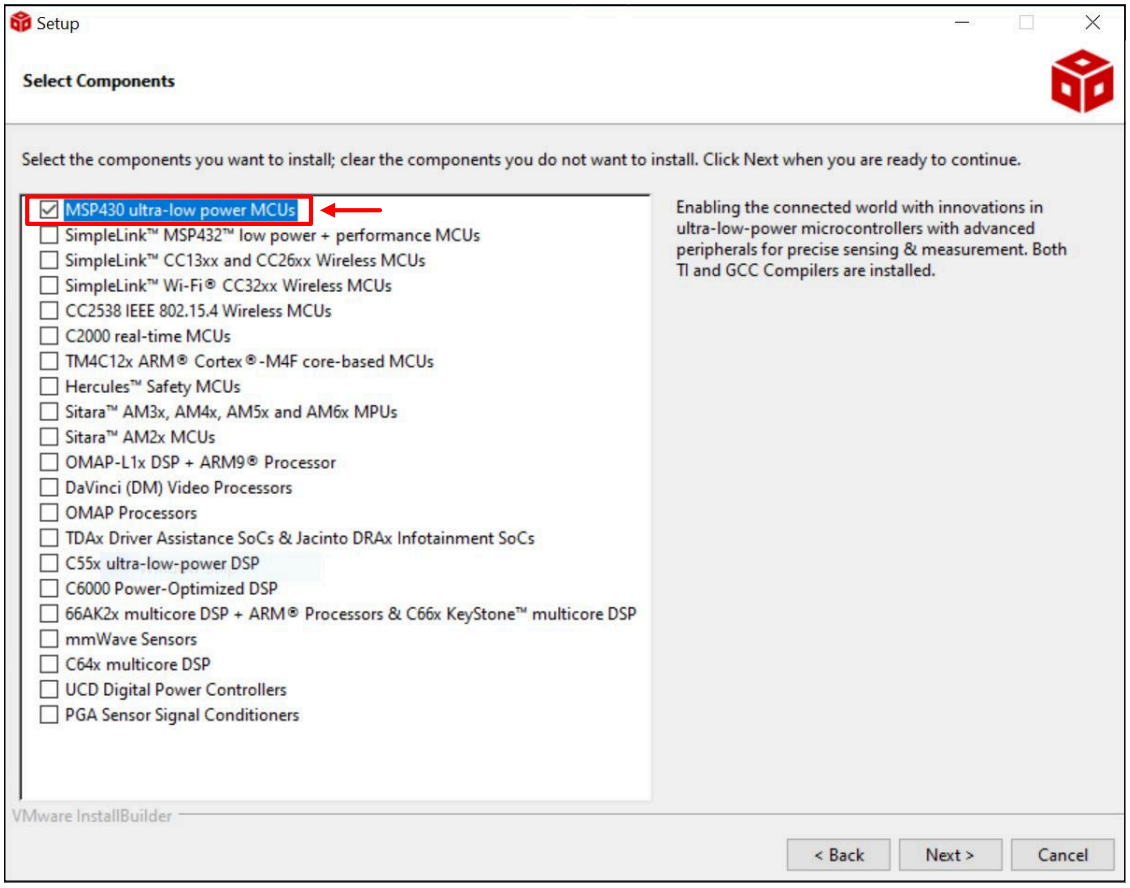


Figure 2-1. Installation Process of Code Composer Studio

2. Download and import sample code.
   a. The link for each EVM is different. However, the sample code in each link is the same except for the file `led_driver.h` that is setup for the matching EVM.
      i. TPS929120EVM: TPS92912XQ1-SW-F5529
      ii. TPS929160EVM: TPS929160Q1-F5529-SW
      iii. TPS929240EVM: TPS929240Q1-F5529-SW
3. Importing the Code Composer Studio (CCS) project according to the process provided in the link: Importing a CCS Project.
4. Load the program according to the process provided in the link: Building and Running Your Project.
5. (optional) Download the EEPROM configuration tool. This is a handy tool if you 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

3 Hardware Setup

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

- *TPS929120EVM User's Guide*
- *TPS929160EVM User's Guide*
- *TPS929240EVM User's Guide*

The sample code replaces the USB2ANY by the MSP-EXP430F5529LP. There are two methods to connect the MSP-EXP430F5529LP to the EVM:

- UART
- TPS929120CANEVM

For both methods the connections are listed in Table 3-1. In addition, Figure 3-1 depicts the locations on the MSP-EXP430F5529LP. Besides the connections, also switches S1 (cyan) and S2 (green) are used in the sample code that is described in more detail in Section 4.1. On the EVM, the jumpers have to be set correct for each mode. This is described in the device-specific EVM User's Guide.

Table 3-1. Hardware Connections

Interface	Board	UART-RX	UART-TX	+3.3V	GND	+5V
	Color	Blue	Yellow	Violet	Black	Orange
	MSP-EXP430F5529LP	P3.4 (J1-3)	P3.3 (J1-4)	3V3 (J1-1)	GND (J3-22)	+5V (J3-21)
CAN	TPS929120CANEVM	J3-13	J3-14	J3-15	J3-16	J3-3
UART	TPS929120EVM	J3-3	J3-4	J3-5	J3-6	Not required
	TPS929160EVM	J29-3	J29-4	J29-5	J29-6	
	TPS929240EVM	J4-3	J4-4	J4-5	J4-6	

The connections are expected to be handwired.

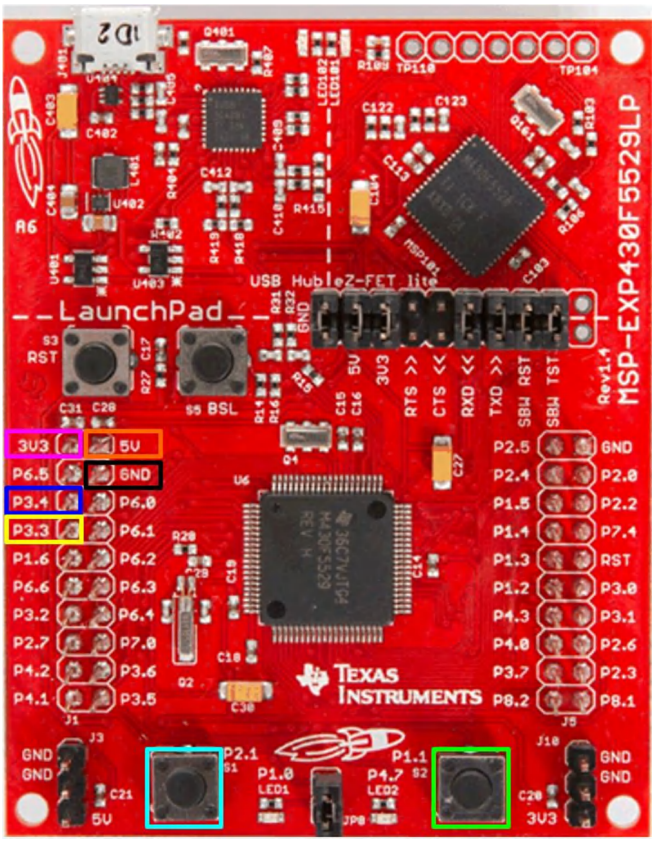


Figure 3-1. Connections on MSP-EXP430F5529LP

4 Sample Code Structure

4.1 Flow Diagram

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

The Setup MCU configures the UART interface and sets it to 750000 bauds. After unlocking the LED driver, it is checked if the animation mode or EEPROM programming mode has been selected. Section 4.2 describes how to change between the modes. During the animation mode, an 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, it is checked if the button S2 on the MSP-EXP430F5529LP was pressed. If it was not pressed, the same LED pattern will be executed again. In case it was pressed, the next LED pattern will be executed until all 6 patterns have been executed and the loop will restart from the first pattern again.

During the EEPROM programming mode both buttons S1 and S2 on MSP-EXP430F5529LP are used as well as 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.

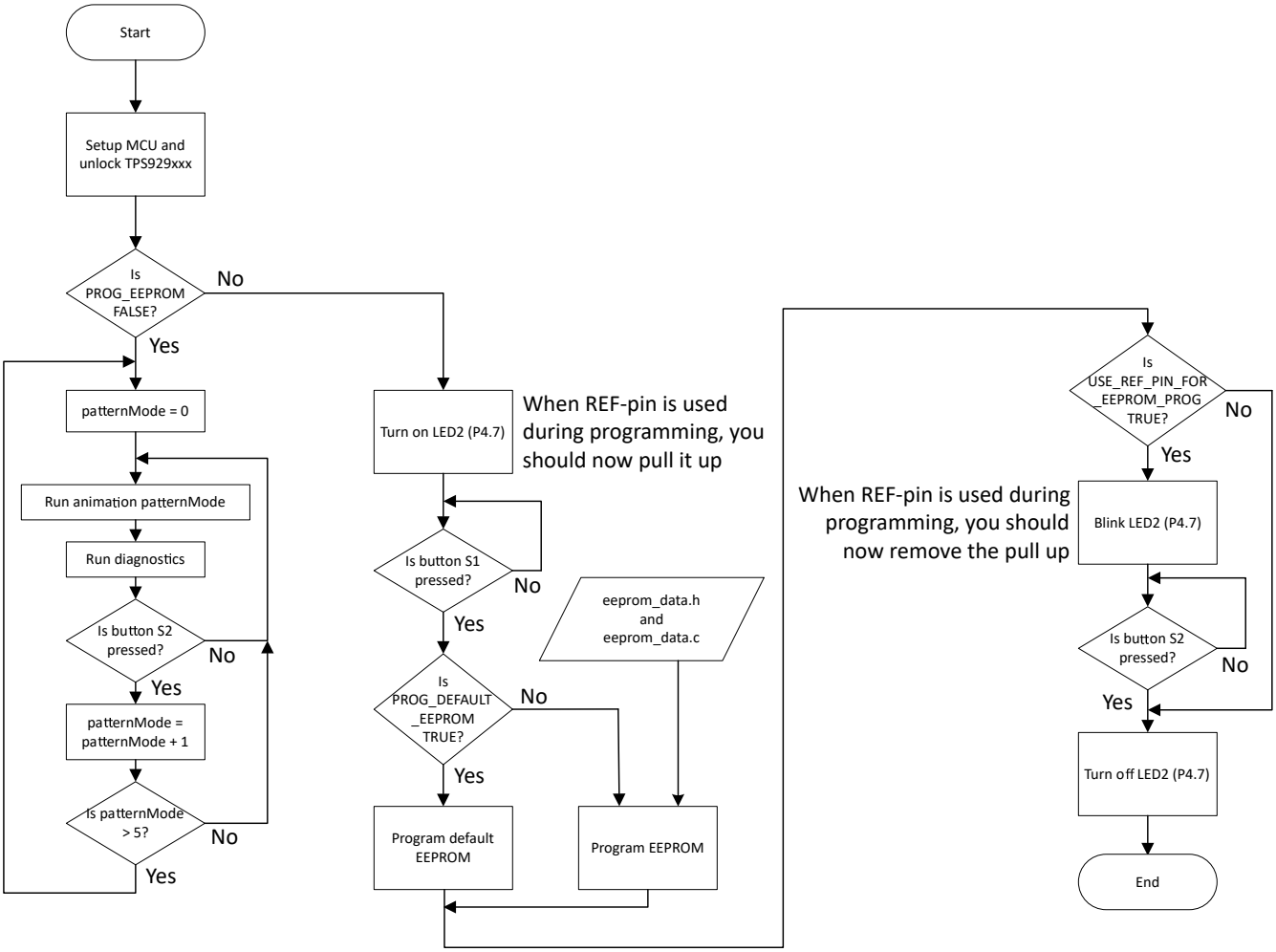


Figure 4-1. Sample Code Flow Diagram

4.2 System Setup

This section describes how the sample code setups 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.

```
#include "TPS929240.h"
```

The code 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 about macros and variables that impact the system setup and their location is listed in Table 4-1.

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

Filename	Macro/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 will 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 it is found that it is 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 could be 16 different addresses. Therefore, the index runs from 0 to 15.

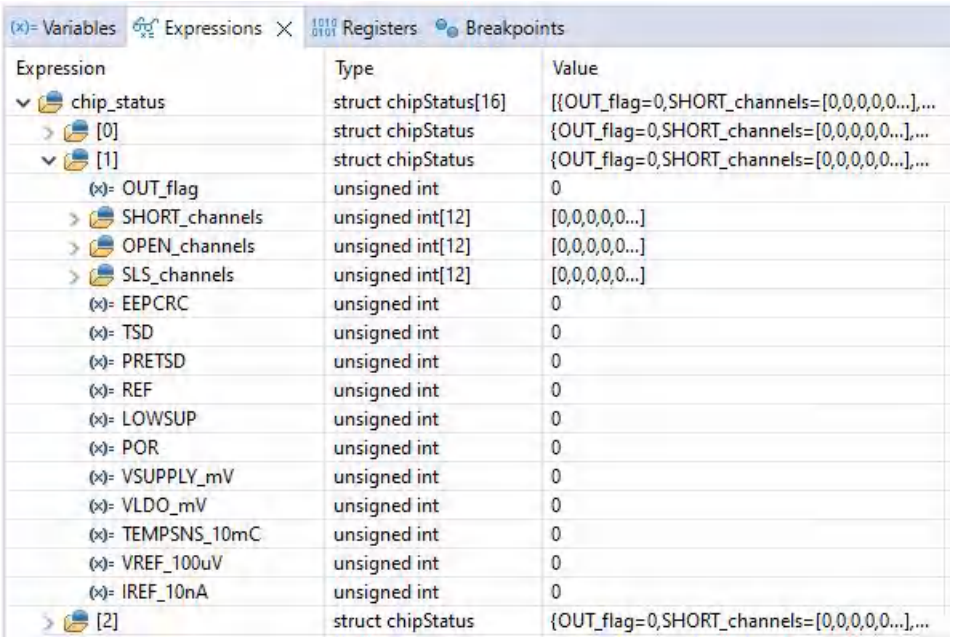
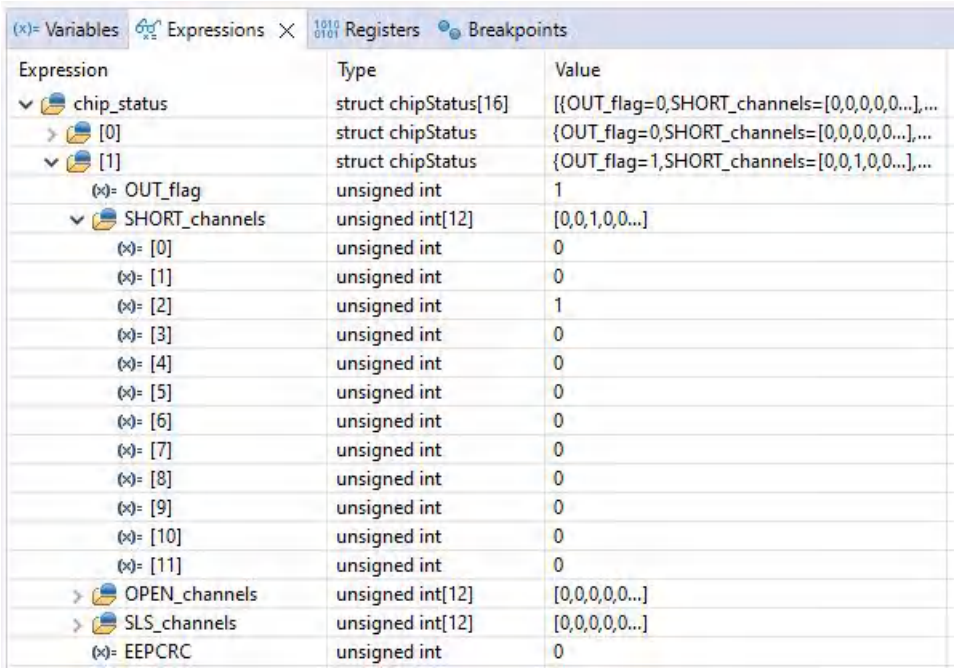


Figure 4-2. Example of Watching Expression chip_status Without Errors for TPS929120-Q1

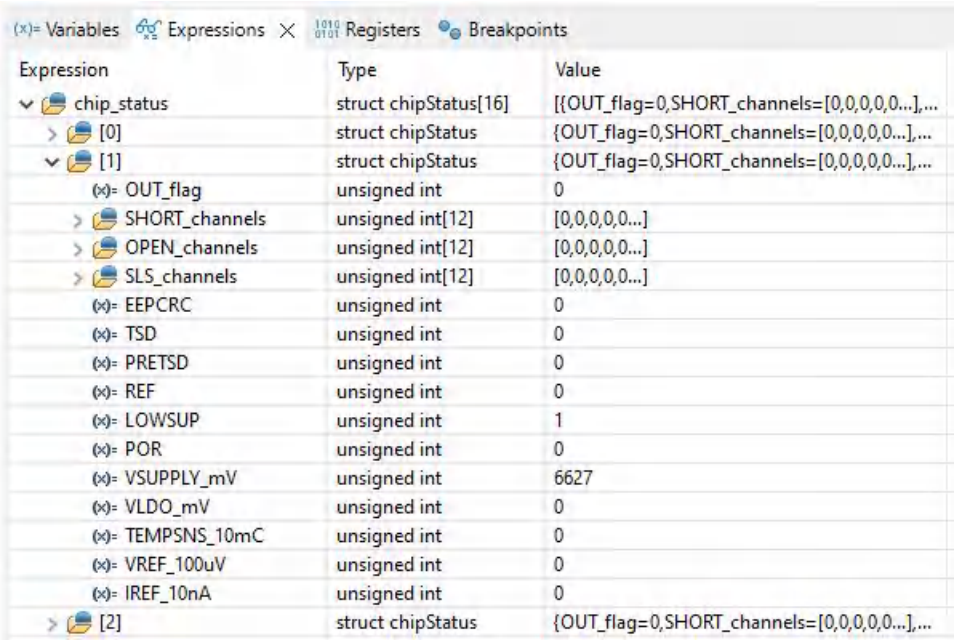
An example with a short is depicted in Figure 4-3. The TPS929120-Q1 has address 0x1 and flag OUT_Flag is set. When the array SHORT_channels is expanded, it can be seen that the short occurs on pin OUT2.



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 int	1
SHORT_channels	unsigned int[12]	[0,0,1,0,0...]
[0]	unsigned int	0
[1]	unsigned int	0
[2]	unsigned int	1
[3]	unsigned int	0
[4]	unsigned int	0
[5]	unsigned int	0
[6]	unsigned int	0
[7]	unsigned int	0
[8]	unsigned int	0
[9]	unsigned int	0
[10]	unsigned int	0
[11]	unsigned int	0
OPEN_channels	unsigned int[12]	[0,0,0,0,0...]
SLS_channels	unsigned int[12]	[0,0,0,0,0...]
EEPCRC	unsigned int	0

Figure 4-3. Example of Watching Expression chip_status with Short Fault for TPS929120-Q1

An example when a low supply warning occurs ($V_{(SUPPLY)}$ < $V_{(ADCLOWSUPTH)}$) in TPS929120-Q1 is depicted in Figure 4-4. The flag LOWSUP has been set for 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 6627 mV.



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 int	0
SHORT_channels	unsigned int[12]	[0,0,0,0,0...]
OPEN_channels	unsigned int[12]	[0,0,0,0,0...]
SLS_channels	unsigned int[12]	[0,0,0,0,0...]
EEPCRC	unsigned int	0
TSD	unsigned int	0
PRETSD	unsigned int	0
REF	unsigned int	0
LOWSUP	unsigned int	1
POR	unsigned int	0
VSUPPLY_mV	unsigned int	6627
VLDO_mV	unsigned int	0
TEMPSNS_10mC	unsigned int	0
VREF_100uV	unsigned int	0
IREF_10nA	unsigned int	0
[2]	struct chipStatus	{OUT_flag=0,SHORT_channels=[0,0,0,0,0...],...

Figure 4-4. Example of Watching Expression chip_status With Low Supply for TPS929120-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 2.

The LED driver IC supports two solutions 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 should be pulled high during programmed. When it is defined as FALSE, the current device address is used. TI recommends to use the current device address.

When the code enters the EEPROM programming routine, it turns on LED2 (P4.7) on the MSP-EXP430F5529LP. When macro *USE_REF_PIN_FOR_EEPROM_PROG* is defined as TRUE, the REF pin should be 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 MSP-EXP430F5529LP should be pressed to start the programming. When the current device address is used, LED2 will turn off once the programming is finished.

When the REF pin is used, LED2 starts to blink after programming is finished. At that time, the pull up on the REF pin should be removed and afterwords the button S2 on MSP-EXP430F5529LP should be pressed. Then LED2 will turn 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)

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