User's Guide

LP589x(-Q1)/TLC698x Sample Code

ABSTRACT

This document describes the preparation and usage of the sample code for the LP589x(-Q1)/TLC698x device family when paired with a LAUCHXL-F280039C. Following the instructions provided for setup, the installed code lights up the LEDs on the EVM. The LP5890/1/2(-Q1) devices can be paired with the LP5899(-Q1) to use a standard SPI module of the MCU to communicate to the LED drivers. Moreover, the TLC6983/4 devices can be paired with the TLC6989 to use a standard SPI module of the MCU to communicate to the LED drivers.

Table of Contents

List of Figures

List of Tables

Trademarks

LaunchPad™ and Code Composer Studio™ are trademarks of Texas Instruments.
All trademarks are the property of their respective owners.

1 Introduction

The sample code showcases the ability to light up the LEDs on the LP5891Q1EVM, LP5891EVM, LP5890EVM, TLC6984EVM, and TLC6983EVM. 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 simple test. The animation mode is selected by default. Section 3.3 describes how to change between the modes. In the animation mode two frames are used to scroll left, right, up, and down and to fade in and fade out according to a predefined sequence. The first frame is a Texas Instruments logo of 32x32 RGB pixels and the second frame a rainbow pattern of 48x32 RGB pixels. This means that not always the full frame is shown on the LED display of the EVM. Examples of this can be seen in Section 3.5. It is outside the scope of this document to explain how the frames in the sample code are generated.

In the simple test mode, the user can use some predefined APIs to light up the LED board, perform diagnostics, or build custom Continuous Clock Serial Interface (CCSI) commands. The sample code comes with turning on all RGB LEDs which results in a white display. In addition, every frame, diagnostics is executed. For more information about diagnostics see Section 3.4.

The LP5891Q1EVM, LP5891EVM, and LP5890EVM can be paired with LP5899DYYEVM. Section 3.3 describes how to enable the pairing. After enabling, the LAUCHXL-F280039C's SPI is used to communicate to LP5899DYYEVM instead of the LAUCHXL-F280039C's CLB (Configurable Logic Block) to communicate to LP589X(Q1)EVM. Moreover, additional diagnostics is enabled.

The predefined APIs in the sample code automatically adjust to the specified system. More detail about the system specification can be found in Section 3.3.

2 Software Setup

To set up the software for the TMS320F280039C LaunchPad™, please 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 12.7.0).
   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 "C2000 real-time MCUs" in "Select Components", as is marked with red box in Figure 2-1.

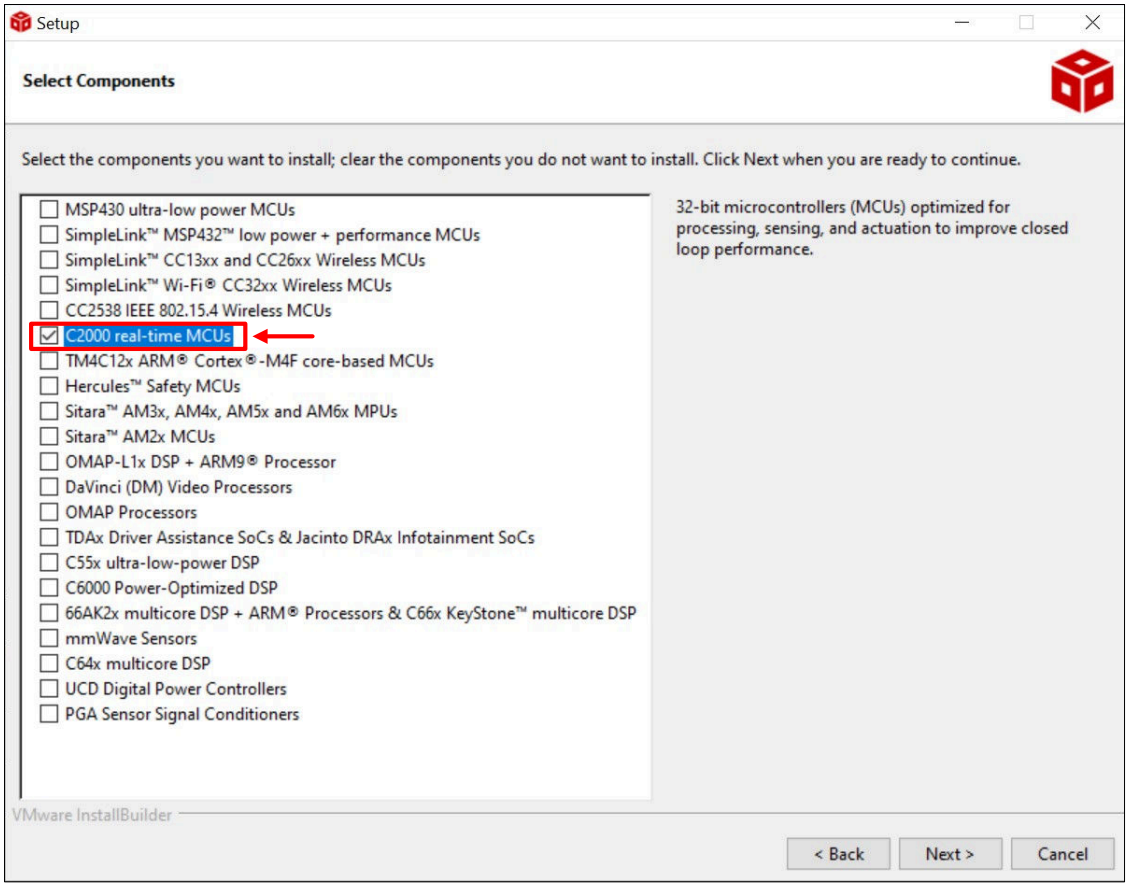

Figure 2-1. Installation Process of Code Composer Studio

2. Download and install C2000Ware (Version 5.01.00.00).
   a. To verify the installation is correct, open the Code Composer Studio software. Click "Windows" from the top menu bar. Then click "Preferences" from the drop-down list. The "Preferences" window will show. From the left side bar, select "Code Composer Studio" -> "Products".
   b. Make sure that there is "C2000Ware 5.1.0.00" and "SysConfig" (SysConfig should be installed automatically when you install C2000Ware) under the "Discovered products", as is marked in Figure 2-2. If there is no "C2000Ware 5.1.0.00" or "SysConfig" appearing, you may need to click "Refresh" on the right side of the "Preference" window. If still unavailable, check whether the installation is correct. A standalone desktop version of SysConfig can be found at SYSCONFIG System configuration tool.

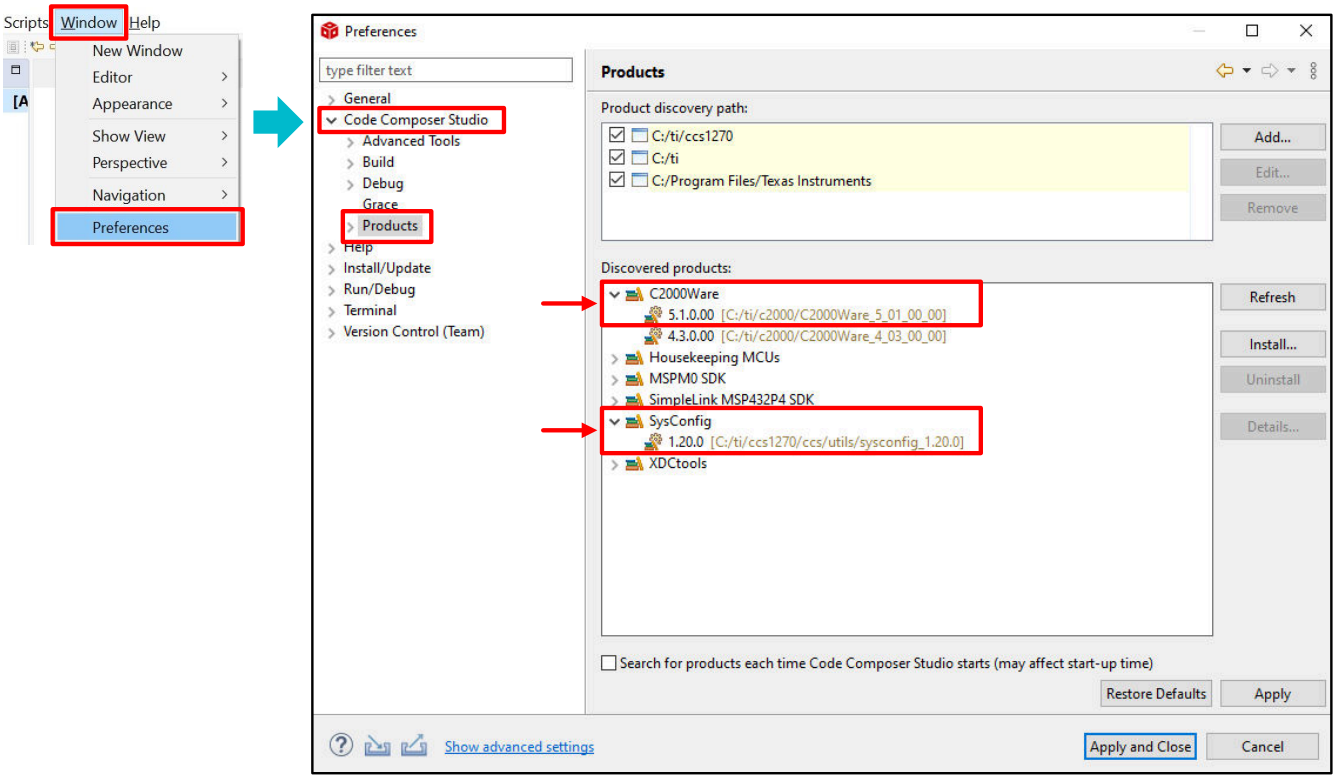

Figure 2-2. Verification of C2000Ware 5.1.0.00 Installation

3. 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 files `led_driver.h` and `system_info.h` which is setup for the matching EVM.
      i. LP5891Q1EVM: LP5891Q1EVM-F280039C-SW
      ii. LP5891EVM: LP5891Q1EVM-F280039C-SW
      iii. LP5890EVM: LP5890EVM-F280039C-SW
      iv. TLC6984EVM: TLC6984EVM-F280039C-SW
      v. TLC6983EVM: TLC6983EVM-F280039C-SW
      vi. LP5899DYYEVM paired with LP5891(Q1)EVM: LP5899DYYEVM-LP5891-F280039C-SW
      vii. LP5899DYYEVM paired with LP5890EVM: LP5899DYYEVM-LP5890-F280039C-SW
   b. 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 register map generation tool. This is a handy tool if you want to further configure the registers.
   a. LP5891-Q1: LP5891 Registers Map Generation Tool
   b. LP5891: LP5891 Registers Map Generation Tool
   c. LP5890: LP5890 Registers Map Generation Tool
   d. TLC6984: TLC6984 Registers Map Generation Tool
   e. TLC6983: TLC6983 Registers Map Generation Tool

3 Sample Code Structure

3.1 Design Parameters

The LED matrix display design parameters used for the different EVMs are listed in Table 3-1 .

Table 3-1. Design Parameters EVMs

Design Parameter	LP589x	TLC698x
Display module size	16 × 16 RGB LEDs	32 × 32 RGB LEDs
Frame rate	60Hz	25Hz
Refresh rate	7680Hz	2400Hz
PWM resolution	16 bits	16 bits
Cascaded devices	1	2
SCLK frequency	10MHz	10MHz
GCLK frequency	80MHz	120MHz

3.2 Flow Diagram

Figure 3-1 depicts the high level flow in the sample code.

During the Setup MCU, several macros defined in file `system_info.h` determine if SPI is used to communicate to the augmented connectivity device or the Configurable Logic Block (CLB) to communicate to the LED drivers using CCSI. When SPI is used for communication, the augmented connectivity device is initialized for the CCSI data rate, and FIFO levels.

The `FC_settings.h` file can automatically be generated by the Registers Map Generation Tool mentioned in Section 2. Not all register settings are coming from this file. The number of scan lines (field SCAN_NUM in register FC0) and the number of cascaded devices (field CHIP_NUM in register FC0) are coming from the file `system_info.h`. This file is described in more detail in Section 3.3.

The flow diagram also shows the files `frames.c` and `frames.h` which contain the 2 frames used during the animation mode. It is outside the scope of this document to explain how these frames are generated.

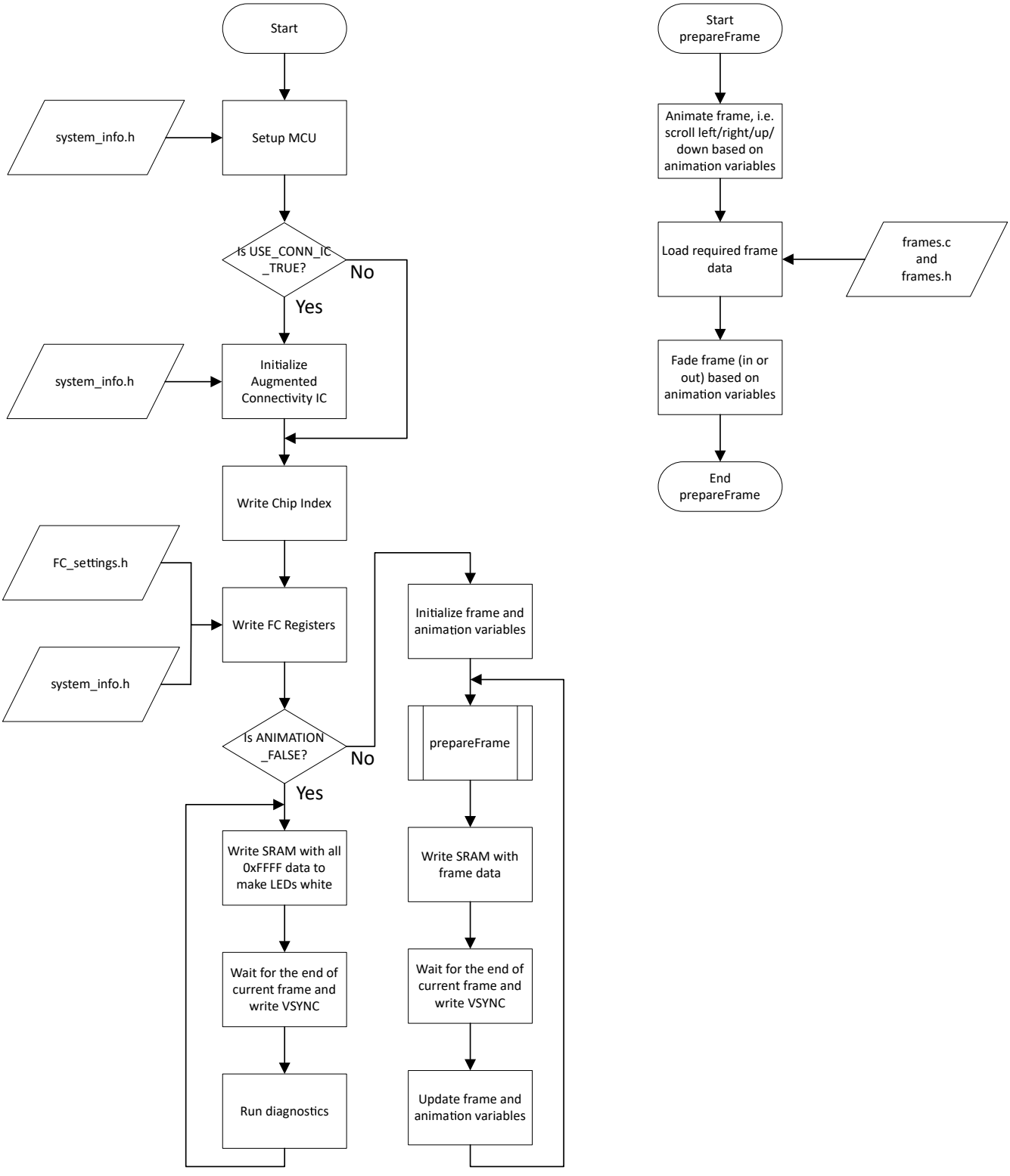

Figure 3-1. Sample Code Flow Diagram

3.3 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 "LP5891.h"
```

The code supports:

- LP5892
- LP5891
- LP5890
- TLC6984
- TLC6983

Note that for the "Q1" devices, this is not added and only the base product name is used.

A summary about macros and variables that impact the system setup and their location is listed in Table 3-2.

Table 3-2. Summary of Macro and Variable Names Per File

Filename	Macro/Variable name	Description
system_info.h	*SCLK_FREQ_IN_HZ*	SCLK frequency
	USE_CONN_IC	Selection if augmented connectivity device LP5899(-Q1)/TLC6989 is used
	SPI_FREQ_IN_HZ	When augmented connectivity device is used, this is the SPI frequency between MCU and connectivity device
	RUN_MODE	Selection between different code run mode. Supported options are *ANIMATION* and *SIMPLE_TEST*
	TOTAL_SCAN_LINES	Number of scan lines
	CCSI_BUS_NUM	Number of CCSI busses
	CASCADED_UNITS_CCSI1	Device count in CCSI bus 1
	CASCADED_UNITS_CCSI2	Device count in CCSI bus 2
	MONOCHROMATIC	Selection between RGB and single color display
system_info.c	*FRAME_RATE*	Interval of VSYNC commands

Within the file `system_info.c` the frame rate is specified. The frame rate is specified in Hz (frames per second).

```
const uint16_t FRAME_RATE = 60;      // 16.67ms = 60 Hz frames-per-second
```

The minimum supported frame rate is 1Hz.

File `system_info.h` includes several system definitions.

```
// When TLC6989 or LP5899 is used this should be set to _TRUE
#define USE_CONN_IC           _TRUE
```

Macro *USE_CONN_IC* defines if the LED driver is paired with the augmented connectivity IC. This impacts the MCU's hardware setup which switches automatically between standard SPI when using the connectivity IC or CLB when communicating directly to the LED drivers using CCSI.

```
// Desired SCLK frequency (in case of TLC698x this SCLK frequency is half of this)
// Note: Exact frequency may not be possible
#define SCLK_FREQ_IN_HZ        10000000
```

Macro *SCLK_FREQ_IN_HZ* defines at what data rate the Continuous Clock Serial Interface (CCSI) is running, i.e. the clock frequency of pin SCLK. When the augmented connectivity IC is used, the selected data rate will be the option that is equal or first available option that is higher than the desired data rate. When the augmented connectivity IC is not used, the desired SCLK frequency is an integer divider from the system frequency of the MCU. In both cases, the actual CCSI frequency may differ from the desired specified frequency defined by *SCLK_FREQ_IN_HZ*.

```
// Desired SPI frequency - for TLC6989 or LP5899
// Supported range is 500kHz to 7.5MHz --> For higher frequencies need to enable SPI high speed mode
// Note: Exact frequency may not be possible
#define SPI_FREQ_IN_HZ          7500000
```

Macro *SPI_FREQ_IN_HZ* is only used when the augmented connectivity IC is used and defines the desired SPI frequency. This is an integer divider from the system frequency of the MCU. Therefore, the actual SPI frequency may differ from the desired specified frequency defined by *SPI_FREQ_IN_HZ*.

```
#define RUN_MODE                ANIMATION
#define MONOCHROMATIC           _FALSE
```

Macro *RUN_MODE* determines the run mode of the code. The supported run modes are animation and simple test mode.

The EVMs all use RGB LEDs. Therefore, macro *MONOCHROMATIC* is defined as _FALSE. The sample code does support systems using single color LEDs, e.g. only red LEDs. In those cases, the macro *MONOCHROMATIC* should be defined as _TRUE. This automatically changes the frame data structure, the animation algorithms, and APIs.

The following code block shows macros which impact the register settings.

```
#define TOTAL_SCAN_LINES        16
#define CASCADED_UNITS_CCSI1    1
```

Macro *TOTAL_SCAN_LINES* defines the number of scan lines used in the system and directly impacts the field SCAN_NUM in register FC0. For the LP5891Q1EVM, LP5891EVM, and LP5890EVM there are 16 scan lines. For the TLC6983EVM and TLC6984EVM there are 32 scan lines.

Macro *CASCADED_UNITS_CCSI1* defines the number of cascaded devices in the system and directly impacts the field CHIP_NUM is register FC0. For the LP5891Q1EVM, LP5891EVM, and LP5890EVM there is only 1 device cascaded. When the user cascades more of these EVMs with the available connectors, this macro has to be updated.

For the TLC6983EVM and TLC6984EVM there are two cascaded devices on one EVM.

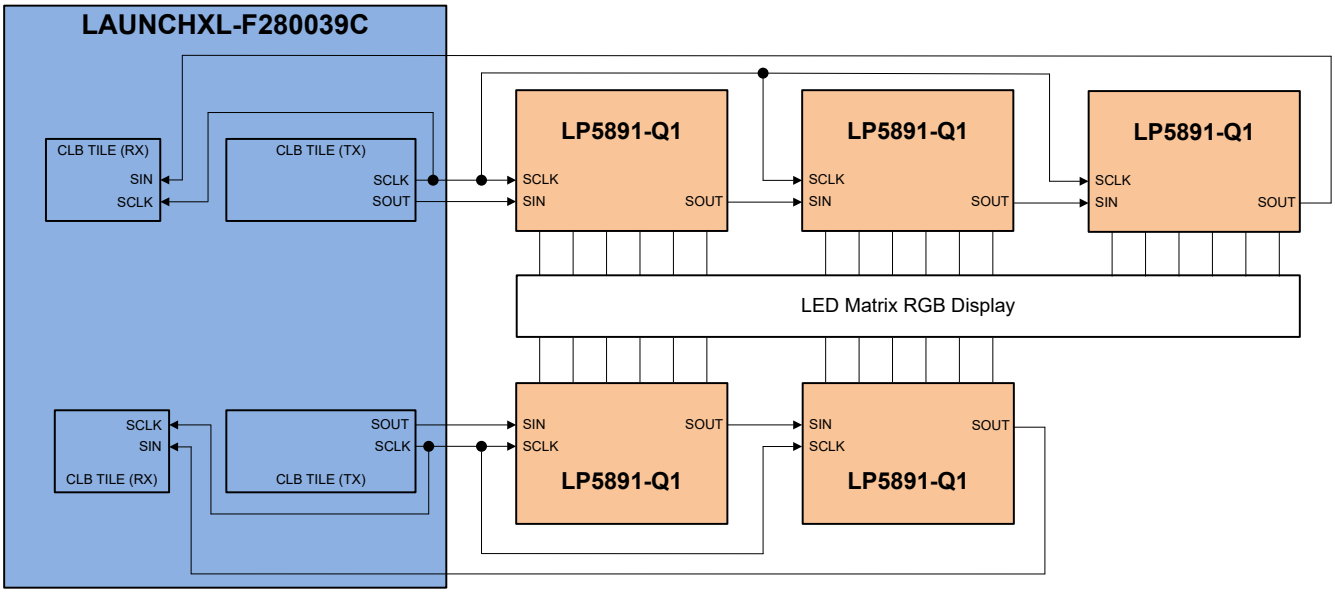

Figure 3-2. Example System Using 2 CCSI Chains Without Augmented Connectivity IC

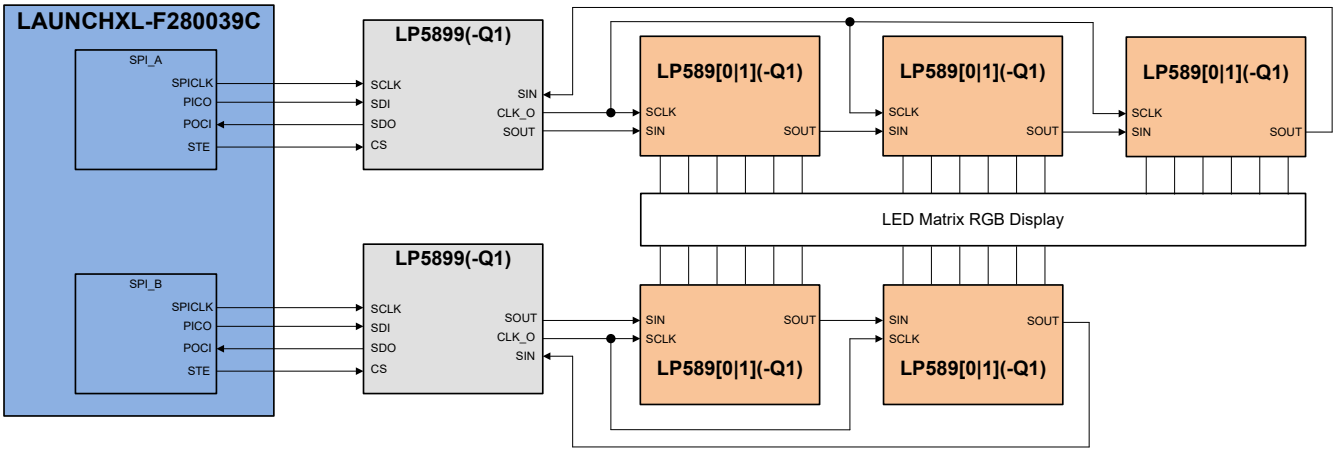

Figure 3-3. Example System Using 2 CCSI Chains With Augmented Connectivity IC

The sample code supports up to 2 CCSI daisy chains. To illustrate the support of 2 CCSI daisy chains, examples are depicted in Figure 3-2 and Figure 3-3. The first figure depicts the example without using the augmented connectivity IC and the second image with using the augmented connectivity IC. For both, the number of actual used chains is defined by macro *CCSI_BUS_NUM* in file `system_info.h`.

```
// Total CCSI buses supported
#define CCSI_BUS_NUM            2
```

Each chain can have a different number of cascaded devices. Therefore, besides macro *CASCADED_UNITS_CCSI1*, there is also macro *CASCADED_UNITS_CCSI2* in file `system_info.h`. In the examples, CCSI chain 1 has 3 cascaded devices and CCSI chain 2 has 2 cascaded devices.

```
#define CASCADED_UNITS_CCSI1    3
#define CASCADED_UNITS_CCSI2    2
```

3.4 Diagnostics

The sample code provides an API to detect which LED drivers have LED open (LOD) and which LED drivers have LED short (LSD) faults. In addition, when the sample code is paired with LP5899DYYEVM, the faults detected by the LP5899 augmented connectivity device are read. Within the `TLC698x_LP589x_APIs.h` file the prototype of the API is defined.

```
void LED_Update_Chip_Status(void);
```

This API updates the variable *chip_status* which is defined in `system_info.h`.

```
struct ccsiBusStatus {
    uint16_t LSD;              // LED Short Detection
    uint16_t LOD;              // LED Open Detection
    volatile outChannel LOD_channels[TOTAL_SCAN_LINES];
    volatile outChannel LSD_channels[TOTAL_SCAN_LINES];
};

struct chipStatus {
    struct ccsiBusStatus busStatus[MAX_CASCADED_UNITS];
#ifdef _TLC6989_LP5899
    uint16_t ERR;                  // LP5899/TLC6989 Global error flag
    uint16_t POR;                  // Power-On-Reset flag
    uint16_t OSC;                  // Internal oscillator flag
    uint16_t OTP_CRC;              // OTP CRC flag
    uint16_t DEV_STATE;            // Device state
    uint16_t SPI_CRC;              // SPI CRC error
    uint16_t SPI_REG_WRITE;        // SPI register write error
    uint16_t SPI;                  // SPI error
    uint16_t SPI_CS;               // SPI Chip Select (CS) pin error
    uint16_t SPI_TIMEOUT;          // SPI reset timeout error
    uint16_t SRST;                 // Softreset error
    uint16_t DRDY_STATUS;          // DRDY pin status
    uint16_t RXFF;                 // Receive FIFO error
    uint16_t RXFFSED;              // Receive FIFO single error detection
    uint16_t RXFFUVF;              // Receive FIFO underflow
    uint16_t RXFFOVF;              // Receive FIFO overflow
    uint16_t TXFF;                 // Transmit FIFO error
    uint16_t TXFFSED;              // Transmit FIFO single error detection
    uint16_t TXFFUVF;              // Transmit FIFO underflow
    uint16_t TXFFOVF;              // Transmit FIFO overflow
    uint16_t CCSI;                 // CCSI interface error
    uint16_t CCSI_SIN;             // CCSI missing toggling on SIN error
    uint16_t CCSI_CRC;             // CCSI data CRC error
    uint16_t CCSI_CHECK_BIT;       // CCSI check bit error
    uint16_t CCSI_CMD_QUEUE_OVF;   // CCSI command queue overflow
#endif
};

// For diagnostics
extern struct chipStatus chip_status[CCSI_BUS_NUM];
```

Note that in the sample code the diagnostics is only executed during simple test mode and not during animation mode.

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 for the LP5891Q1EVM without any error is depicted in Figure 3-4. The first index of variable *chip_status* is the CCSI chain index. On the EVM only 1 chain is used. For the flags of the LED drivers, the busStatus variable is used which has an index for each LED driver in the chain.

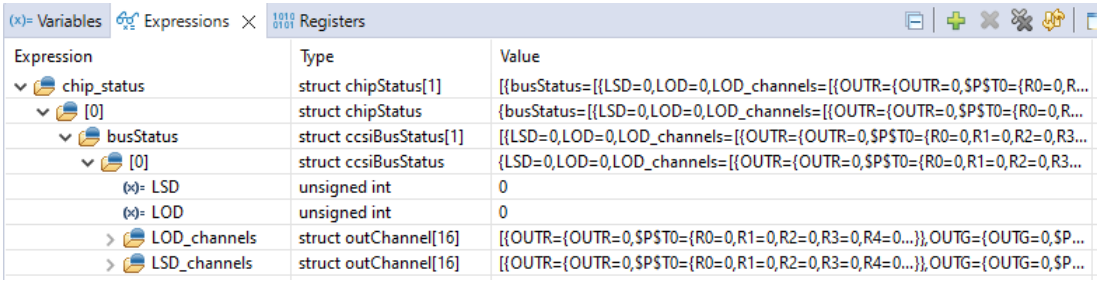

Figure 3-4. Example of Watching Expression chip_status Without Errors

Figure 3-5 depicts an expanded view of the *chip_status* variable. For each LED driver device in the chain, the LSD and LOD are shown. In addition, for the LSD and LOD faults the actual line and output channel which has the fault can be found. The LP5891Q1EVM uses 16 scan lines. Therefore, the indices for array LOD_channels run from 0 to 15.

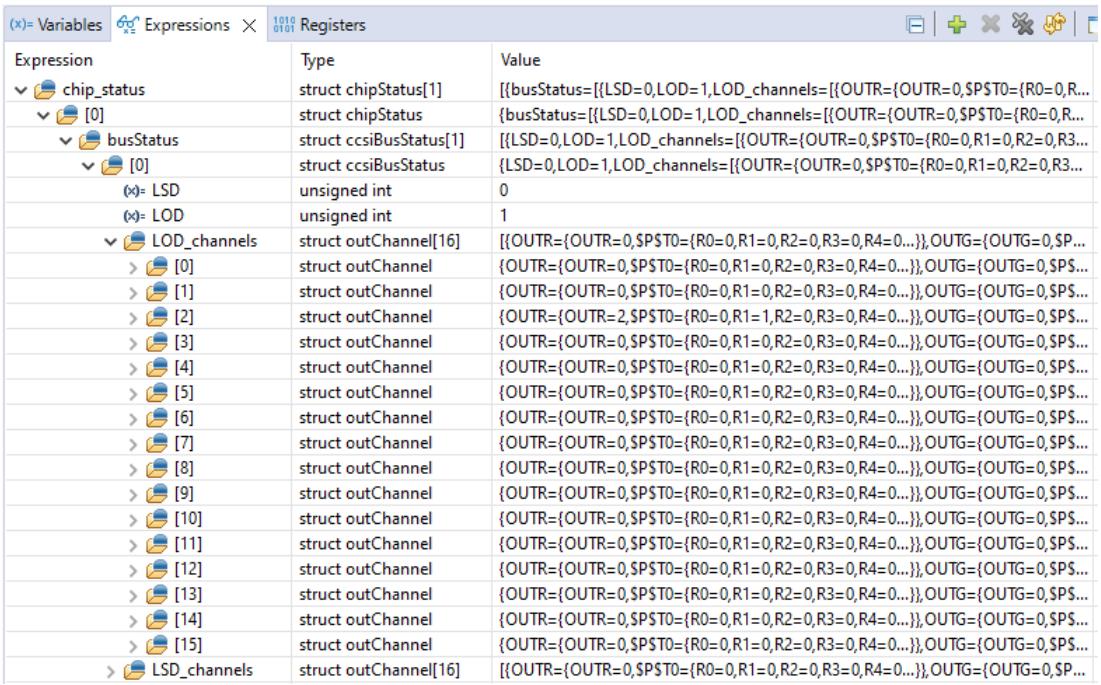

Figure 3-5. Example Showing Expanded Scan Lines of Expression chip_status With LOD

An example of an LOD fault is depicted in Figure 3-6. In this example chip index 0 has an LOD fault. When array LOD_channels is expanded, it shows that the fault occurs on line with index 2. When that index is expanded, it shows that the fault happens on pin R1.

(x)= Variables | Expressions | Registers

Expression	Type	Value
chip_status	struct chipStatus[1]	[{busStatus=[{LSD=0,LOD=1,LOD_channels=[{OUTR={OUTR=0,PT0={R0=0,R...
[0]	struct chipStatus	{busStatus=[{LSD=0,LOD=1,LOD_channels=[{OUTR={OUTR=0,PT0={R0=0,R...
busStatus	struct ccsiBusStatus[1]	[{LSD=0,LOD=1,LOD_channels=[{OUTR={OUTR=0,PT0={R0=0,R1=0,R2=0,R3...
[0]	struct ccsiBusStatus	{LSD=0,LOD=1,LOD_channels=[{OUTR={OUTR=0,PT0={R0=0,R1=0,R2=0,R3...
LSD	unsigned int	0
LOD	unsigned int	1
LOD_channels	struct outChannel[16]	[{OUTR={OUTR=0,PT0={R0=0,R1=0,R2=0,R3=0,R4=0...}},OUTG={OUTG=0,$P...
[0]	struct outChannel	{OUTR={OUTR=0,PT0={R0=0,R1=0,R2=0,R3=0,R4=0...}},OUTG={OUTG=0,P...
[1]	struct outChannel	{OUTR={OUTR=0,PT0={R0=0,R1=0,R2=0,R3=0,R4=0...}},OUTG={OUTG=0,P...
[2]	struct outChannel	{OUTR={OUTR=2,PT0={R0=0,R1=1,R2=0,R3=0,R4=0...}},OUTG={OUTG=0,P...
OUTR	union <unnamed>	{OUTR=2,PT0={R0=0,R1=1,R2=0,R3=0,R4=0...}}
OUTR	unsigned int	2
PT0	struct <unnamed>	{R0=0,R1=1,R2=0,R3=0,R4=0...}
R0	unsigned int : 1	0
R1	unsigned int : 1	1
R2	unsigned int : 1	0
R3	unsigned int : 1	0
R4	unsigned int : 1	0
R5	unsigned int : 1	0
R6	unsigned int : 1	0
R7	unsigned int : 1	0
R8	unsigned int : 1	0
R9	unsigned int : 1	0
R10	unsigned int : 1	0
R11	unsigned int : 1	0
R12	unsigned int : 1	0
R13	unsigned int : 1	0
R14	unsigned int : 1	0
R15	unsigned int : 1	0
OUTG	union <unnamed>	{OUTG=0,PT1={G0=0,G1=0,G2=0,G3=0,G4=0...}}
OUTB	union <unnamed>	{OUTB=0,PT2={B0=0,B1=0,B2=0,B3=0,B4=0...}}
[3]	struct outChannel	{OUTR={OUTR=0,PT0={R0=0,R1=0,R2=0,R3=0,R4=0...}},OUTG={OUTG=0,P...

Figure 3-6. Example Showing Expanded Channels of Expression chip_status With LOD

When the LP5899DYYEVM is used to pair with LP589x(-Q1), the *chip_status* variable also shows the error flags of the augemented connectivity IC. An example without any fault is depicted in Figure 3-7. For each CCSI chain (first index of *chip_status*) the error flags are listed. Note that DRDY_STATUS is one, because when LP5899 has data that is ready to be read, this value becomes zero.

(x)= Variables | Expressions | Registers

Expression	Type	Value
chip_status	struct chipStatus[1]	[{busStatus=[{LSD=0,LOD=0,LOD_channels=[{OUTR={OUTR=0,PT0={R0=0,R...
[0]	struct chipStatus	{busStatus=[{LSD=0,LOD=0,LOD_channels=[{OUTR={OUTR=0,PT0={R0=0,R...
busStatus	struct ccsiBusStatus[1]	[{LSD=0,LOD=0,LOD_channels=[{OUTR={OUTR=0,PT0={R0=0,R1=0,R2=0,R3...
ERR	unsigned int	0
POR	unsigned int	0
OSC	unsigned int	0
OTP_CRC	unsigned int	0
DEV_STATE	unsigned int	0
SPI_CRC	unsigned int	0
SPI_REG_WRITE	unsigned int	0
SPI	unsigned int	0
SPI_CS	unsigned int	0
SPI_TIMEOUT	unsigned int	0
SRST	unsigned int	0
DRDY_STATUS	unsigned int	1
RXFF	unsigned int	0
RXFFSED	unsigned int	0
RXFFUVF	unsigned int	0
RXFFOVF	unsigned int	0
TXFF	unsigned int	0
TXFFSED	unsigned int	0
TXFFUVF	unsigned int	0
TXFFOVF	unsigned int	0
CCSI	unsigned int	0
CCSI_SIN	unsigned int	0
CCSI_CRC	unsigned int	0
CCSI_CHECK_BIT	unsigned int	0
CCSI_CMD_QUEUE_OVF	unsigned int	0

Figure 3-7. Example of chip_status Without Errors When Paired With LP5899

An example of a CCSI fault is depicted in Figure 3-8. In this example, the SIN pin is stuck-at high while CCSI transmission is continued, which also results in an overflow for the CCSI command queue.

Variables | Expressions | Registers

Expression	Type	Value
chip_status	struct chipStatus[1]	[{busStatus=[{LSD=0,LOD=0,LOD_channels=[{OUTR={OUTR=0,PT0={R0=0,R...
[0]	struct chipStatus	{busStatus=[{LSD=0,LOD=0,LOD_channels=[{OUTR={OUTR=0,PT0={R0=0,R...
busStatus	struct ccsiBusStatus[1]	[{LSD=0,LOD=0,LOD_channels=[{OUTR={OUTR=0,PT0={R0=0,R1=0,R2=0,R3...
ERR	unsigned int	1
POR	unsigned int	0
OSC	unsigned int	0
OTP_CRC	unsigned int	0
DEV_STATE	unsigned int	0
SPI_CRC	unsigned int	0
SPI_REG_WRITE	unsigned int	0
SPI	unsigned int	0
SPI_CS	unsigned int	0
SPI_TIMEOUT	unsigned int	0
SRST	unsigned int	0
DRDY_STATUS	unsigned int	1
RXFF	unsigned int	0
RXFFSED	unsigned int	0
RXFFUVF	unsigned int	0
RXFFOVF	unsigned int	0
TXFF	unsigned int	0
TXFFSED	unsigned int	0
TXFFUVF	unsigned int	0
TXFFOVF	unsigned int	0
CCSI	unsigned int	1
CCSI_SIN	unsigned int	1
CCSI_CRC	unsigned int	0
CCSI_CHECK_BIT	unsigned int	0
CCSI_CMD_QUEUE_OVF	unsigned int	1

Figure 3-8. Example Showing Expression chip_status With CCSI Fault When Paired With LP5899

3.5 Demo

In this section, examples of the LED demo are presented. Figure 3-9 depicts the LP5891Q1EVM running a demo.

Figure 3-10 depicts the TLC6984EVM running a demo.

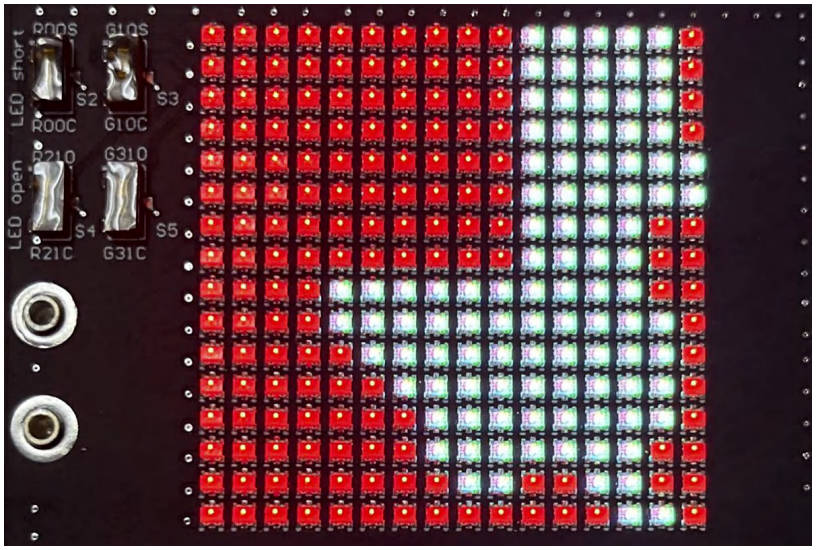

Figure 3-9. LP5891Q1EVM Demo Example

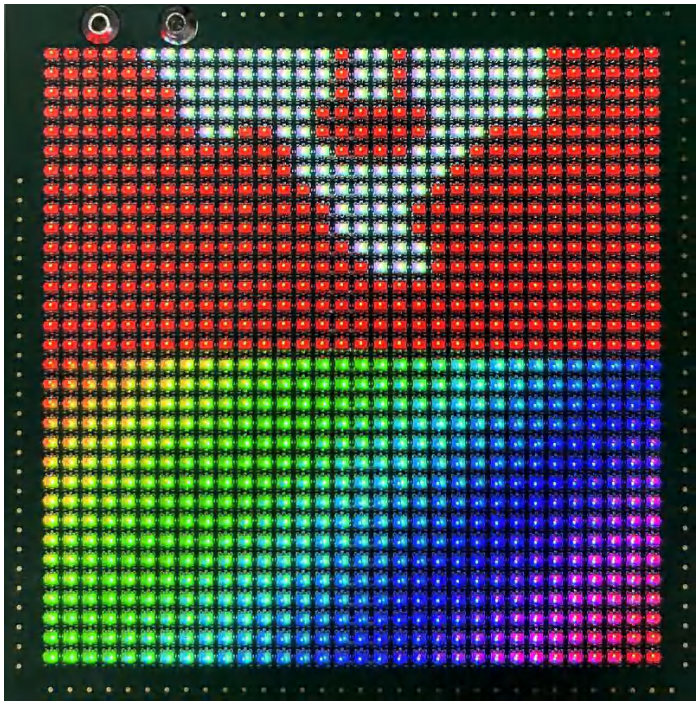

Figure 3-10. TLC6984EVM Demo Example

4 Revision History

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

Changes from Revision * (May 2023) to Revision A (September 2024)	Page
• Updated throughout the document for the whole LP589x/TLC698x device family.	1
• Updated Section 2.	3
• Updated Section 3.2.	5
• Updated Section 3.3.	7
• Updated Section 3.4.	10

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