Application Note

MSPM33C における LVGL の使用

TEXAS INSTRUMENTS

概要

Light and Versatile グラフィック ライブラリ (LVGL) は、MSP M33 ソフトウェア開発キット (SDK) に組み込まれているオープンソースのグラフィック ライブラリです。このソフトウェア ソリューションは、ハードウェアに依存しない高いカスタマイズ性を備えたミドルウェア層を提供します。LVGL と MSPM33 を使用することで、ユーザーは組み込み向けのグラフィカル ユーザー インターフェイス (GUI) を容易に作成でき、MSPM33 マイコンと対応ディスプレイの任意の組み合わせに対して、簡単に実装および移植できます。このアプリケーション ノートでは、いくつかの最終製品 (EE) 向けの GUI を作成するために LVGL を構成する方法を記載するためのいくつかのプログラミング例について説明します。この目的は、ユーザーが LVGL ライブラリを使用して GUI 開発を理解できるようにすることです。

これらのコード例は MSPM33C321A 上でテストされていますが、いずれの例も、任意の MSPM33 デバイスで動作するように容易に適用できます。ほとんどの例では、シリアル ペリフェラル インターフェイス (SPI) またはクワッド シリアル ペリフェラル インターフェイス (QSPI) モジュールを使用して LVGL ピクセル データをディスプレイに送信し、集積回路間通信 (I2C) モジュール、SPI、QSPI モジュールを使用してディスプレイからタッチ入力を読み取ります。この要件は、SPI または QSPI に対応したディスプレイドライバを内蔵し、かつ I2C、SPI、または QSPI に対応したタッチ ドライバを内蔵するディスプレイを選択することで満たすことができます。SPI/ QSPI 信号をデコードできる十分なデジタル入力数を備えたオシロスコープは、デバッグに不可欠です。

目次

図の一覧

商標

1 概要

LVGL は、組込みシステム向けに最適化された、軽量なオープン ソースのグラフィック ライブラリを提供します。高性能レンダリング、柔軟な UI コンポーネント、および広範なハードウェア互換性を備えた LVGL は、リソースに制約のあるデバイス向けのシームレスで最新の GUI 開発を可能にします。テキサス インスツルメンツのお客様は、LVGL の効率的なメモリ

使用、ハードウェア アクセラレーションのサポート、および統合のしやすさからメリットを得られるため、最小限の開発労力で、視覚に訴える高速なインターフェイスを作成できます。

LVGL は、オプションのミドルウェア ライブラリとして MSP M33 SDK に統合されています。LVGL を活用するには、以下の主な考慮事項を考慮する必要があります:

- プロジェクトの設定
- 構成
- 初期化
- グラフィック出力
- グラフィック入力
- グラフィックス更新

このアーキテクチャは図 1-1 で説明されています。

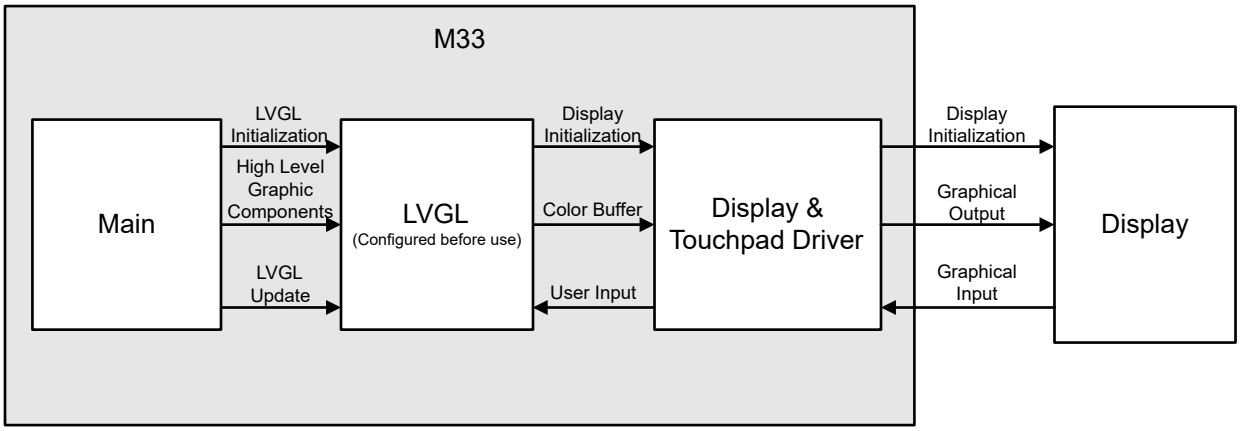

図 1-1. M33 LVGL のアーキテクチャ

1.1 LVGL プロジェクトのセットアップ

LVGL は M33 SDK に統合されていますが、ライブラリはプロジェクトにリンクする必要があります。このリンクを実行するには、次の手順を実行します:

- プロジェクトを右クリックし、「ファイル / フォルダの追加...」を選択します
- 「ファイル / フォルダの追加」ウィンドウで、「+」記号を選択して LVGL ライブラリを追加します
- 「ファイル / フォルダの選択」ウィンドウで、ドロップダウンを「追加するファイルを選択」から「リンクするフォルダを選択」に変更します
- 次に、「パス」の近くにある「...」アイコンをクリックして、SDK LVGL ライブラリのベース パスを選択します。
  - SDK に含まれる LVGL ライブラリは C:\ti\mspm33_sdk_XX\source\third_party\lvgl に配置されています。ここで「XX」は最新の SDK バージョンを示します。
- 「ファイル / フォルダの選択」ウィンドウで「Ok」をクリックします
- 最後に、「ファイル / フォルダの追加」ウィンドウで「Ok」をクリックします
- これで、SDK の LVGL ライブラリがプロジェクトに含まれるはずです

このライブラリには、最も基本的な設定のみが含まれており、低レベル ドライバは含まれていません。特定のアプリケーション向けにライブラリを設定し、特定のディスプレイ用の低レベル ドライバを追加するには、5 つのファイルをプロジェクトに追加し、調整する必要があります:

- lv_conf.h：LVGL のメイン設定ファイル
- lv_port_disp.h：LVGL ディスプレイドライバのヘッダ ファイル
- lv_port_disp.c：LVGL ディスプレイドライバのソース ファイル
- lv_port_indev.h：LVGL 入力デバイスドライバのヘッダ ファイル
- lv_port_indev.c：LVGL 入力デバイスドライバのソース ファイル

1.2 構成

LVGL は高度な設定が可能で、ライブラリの設定は lv_conf.h ファイルで行われます。このファイルには、ライブラリが正しく動作するように更新する必要があるフィールドがいくつかあります。以下のフィールドがあります:

- MY_DISP_HOR_RES:使用するディスプレイに応じて、内部カラー バッファの水平方向のサイズを設定するために使用されます
- MY_DISP_VER_RES:使用するディスプレイに応じて、内部カラー バッファの垂直方向のサイズを設定するために使用されます
- LV_COLOR_DEPTH:使用するディスプレイに応じて、カラー バッファのカラー フォーマットを設定するために使用されます
- LV_MEM_SIZE:オブジェクトやアニメーションを格納するために使用する動的メモリ量を設定するための項目で、通常は 48kB に設定されます
- LV_DRAW_COMPLEX:角丸、円、円弧などの複雑な形状を有効化するために使用されます

アプリケーションのメモリ フットプリントを減らすために調整可能な lv_conf.h セクションもいくつかあります。以下のフィールドがあります:

- フォント:使用していないフォントを無効にして、必要なフラッシュ サイズを小さくします
- ウィジェット:使用しないウィジェット / デモを無効にして、必要なフラッシュサイズを縮小します
- デモ:未使用のデモを無効にして、必要なフラッシュ サイズを小さくします

1.3 初期化

LVGL を用いて組み込み GUI を作成する際には、使用するディスプレイを初期化し、LVGL がそのディスプレイで動作するように設定する必要があります。この初期化および設定は、LVGL グローバル関数 lv_port_disp_init によって行われます。この機能は、次の 3 つのセクションに分かれています:

- 表示の初期化
  - LVGL には、グローバル関数 disp_init が含まれています。この関数は lv_port_disp_init で呼び出され、使用中のディスプレイの初期化シーケンスをトリガするために使用できます。
- カラー バッファのアロケーションとリフレッシュ モードの選択
  - LVGL は、描画された画像のピクセル データを格納するためにカラー バッファを使用します。このバッファである lv_disp_draw_buf_t は、lv_port_disp_init 内で割り当てられます。2 つの代表的なバッファ設定があります:
    - シングル バッファ:1 つのバッファが割り当てられ、lv_disp_draw_buf_init に渡されます。このバッファのサイズは、通常、水平方向の解像度の 10 倍 (ピクセル 10 行分) に設定されます。このバッファには、描画された画像のピクセル データのみが格納されます。また、カラー バッファ内のピクセル情報をディスプレイとの通信に使用される通信インターフェイスへ転送する処理は、CPU が担当します。
    - DMA 付きダブルバッファ:2 つのバッファは同じメモリ量で割り当てられ、lv_disp_draw_buf_init に渡されます。どちらのバッファも、通常 10 行のピクセル サイズです。LVGL は、描画された画像のピクセルデータの一部を 1 つのバッファに格納し、その後、そのデータを DMA が通信インターフェイスへ転送できるようにします。
  - また、LVGL が画像のリフレッシュをどのように行うかを設定するために、2 つのモードがあります:
    - 部分リフレッシュ:パーシャル リフレッシュでは、更新される描画画像の一部のみがカラー バッファに書き込まれます。これはデフォルトではイネーブルです。このモードでは、より高速なリフレッシュと小さなカラーバッファが可能になりますが、ディスプレイがティアリングやバンディングの影響を受けやすい場合、表示に残像やアーティファクトが発生することがあります。
    - フル リフレッシュ:フル リフレッシュでは、レンダリングされた画像全体がカラーバッファに書き込まれます。これは、ディスプレイドライバ lv_disp_drv_t の full_refresh フィールドを設定することで有効になります。このモードでは表示に残るアーティファクトの問題は解消されますが、処理速度は大幅に低下し、画面の解像度と同等サイズのカラーバッファが必要になります。
- ディスプレイドライバ構成
  - LVGL はディスプレイドライバ構造体を提供しており、LVGL がディスプレイを正しく描画できるように、この構造体を設定する必要があります。最初のステップでは、lv_disp_drv_init をドライバの新しいインスタンス lv_disp_drv_t で呼び出して、ディスプレイドライバを初期化します
  - ディスプレイドライバが初期化された後、設定する必要がある主な値は次のとおりです:

    - 水平および垂直解像度：水平および垂直解像度 hor_res および ver_res は、ディスプレイドライバのメンバであり、以前に定義した MY_DISP_HOR_RES および MY_DISP_VER_RES で設定できます
    - フラッシュ コールバック：ディスプレイドライバには、フラッシュ コールバック、flush_cb の場所を設定する必要があります。flush コールバックは、レンダリングされたイメージのピクセルデータが入力されたカラー バッファを LVGL がフラッシュするときに呼び出されるメソッドです。このコールバック メソッドについては、このアプリケーション ノートで詳しく説明します
    - 描画バッファ：ディスプレイドライバは、以前に初期化されたカラー バッファ draw_buf の場所で設定する必要があります
    - リフレッシュ モード：フル リフレッシュ モードを使用する場合、ディスプレイドライバでは full_refresh を 1 に設定する必要があります
  - 最後に、lv_disp_drv_register を呼び出して、ディスプレイドライバを LVGL に登録する必要があります

LVGL は、タッチパッド、マウス、キーパッド、エンコーダ、ボタンなど、複数種類の入力デバイスをサポートしています。これらの入力デバイスはすべて、LVGL を使用して初期化および構成する必要があります。これは、LVGL のグローバル関数 lv_port_indev_init 内で実行され、次の 2 つのセクションに分かれています：

- 入力デバイスの初期化
  - LVGL には、入力デバイスを初期化するために使用できる複数のグローバル関数が用意されています。たとえば、抵抗性タッチまたは静電容量タッチのいずれかを備えたディスプレイを使用する場合、グローバル関数 touchpad_init を呼び出してタッチパッドの初期化を実行できます。
- 入力デバイス ドライバの構成
  - LVGL は入力デバイス ドライバ構造体を提供しており、LVGL が入力を正しく処理できるように、この構造体を設定する必要があります。最初の手順は、新しいドライバ インスタンス lv_indev_drv_t を用意し、lv_indev_drv_init を呼び出して入力デバイス ドライバを初期化することです
  - 入力デバイス ドライバが初期化された後、設定する必要がある主な値は次のとおりです：
    - 入力デバイスのタイプ：使用する入力デバイスの種類は、ポインタ (タッチパッドまたはマウス)、キーパッド、ボタン、エンコーダです
    - 入力読み取りコールバック：入力デバイス ドライバには、入力を読み取るコールバック関数 read_cb の位置を設定する必要があります。入力読み取りコールバックは、LVGL が入力デバイスからの入力を取得する際に呼び出されるメソッドです。このコールバック メソッドについては、このアプリケーション ノートで詳しく説明します
  - 最後に、lv_indev_drv_register を呼び出して、入力デバイス ドライバを LVGL に登録する必要があります

1.4 LVGL 出力

LVGL がカラー バッファをフラッシュする準備ができると、LVGL は設定手順で登録されたフラッシュ コールバック メソッドを呼び出します。フラッシュ コールバックは、カラー バッファが処理され、ディスプレイ固有の通信インターフェイスに送信されます。一般的な処理方法は次のとおりです：

- 個別ピクセル フラッシュ：ピクセル データは通信インターフェイスに 1 つずつ書き込まれ、ディスプレイに送信されます。この方法は最もシンプルですが、CPU による処理時間が必要となるため、最も低速でもあります
- フル バッファ フラッシュ：ピクセル データは、DMA 経由で通信インターフェイスに転送され、ディスプレイに送信されます。この方法は DMA を使用するため、前の方法よりも複雑になりますが、DMA を用いることで、LVGL は 2 つ目のカラー バッファに次の描画画像のデータを書き始めることができます。その結果、この方法は大幅に高速になります。

カラー バッファの処理が完了した後、lv_disp_flush_ready を呼び出して、カラー バッファをフラッシュ可能であることを LVGL に通知する必要があります。

1.5 LVGL 入力

LVGL が入力デバイスからの入力を読み取る準備ができると、LVGL は設定手順で登録された入力読み取りコールバック メソッドを呼び出します。入力読み取りコールバックは、入力デバイスが入力情報のポーリングを受ける場所です。入力デバイスから読み取る際の一般的な処理フローは次のとおりです：

1. 入力デバイスが操作されたかどうかを確認し、その操作内容に応じて入力デバイス ドライバの状態を更新します

2. 入力デバイスから入力情報を取得します。これは通常、タッチパッドの場合は x/y 座標、キーパッドの場合はキー番号となります。取得した入力情報を用いて、入力デバイスの種類に対応する入力デバイスドライバの各フィールドを設定します

1.6 LVGL の更新

LVGL がアニメーションやその他のタスクを正しく実行するためには、LVGL にはシステム ティック、周期的なタイマ トリガ、そしてスリープ管理が必要です。これらの要件の詳細情報は、次のとおりです:

- システム ティック:LVGL は経過時間を通知する必要があります。これは、経過した時間をミリ秒単位で指定して lv_tick_inc 関数を周期的に呼び出すことで実現されます。これは、タイマ割り込み内、またはメインの while ループ内のいずれかで実行できます。
- 定期的なタイマ トリガ:LVGL には、タイマとタスクを処理するための専用 CPU 時間が必要です。これは、システム ティックと同様の方法で lv_timer_handler を定期的に呼び出すことによって行われます。
- スリープ管理:場合によっては、マイコンがスリープ モードに入る際に、そのことを LVGL に通知する必要があります。タイマ割り込みを使用して lv_tick_inc または lv_timer_handler を呼び出す場合、この割り込みを停止して、マイコンがウェークアップしたときに LVGL が同じ状態に戻るようにする必要があります。

2 LVGL の例

次に示す LVGL のサンプルでは、LP-MSPM33C321A を TFT ディスプレイに接続し、ST7796 ディスプレイドライバと DFT6636U 静電容量式タッチ ドライバを使用しています。ST7796 ディスプレイドライバは SPI を介して LP-MSPM33C321A と通信し、DFT6636U タッチ ドライバは I2C インターフェイスを介して LP-MSPM33C321A と通信します。

2.1 ハードウェア接続

LP-MSPM33C321A と TFT ディスプレイとの間の接続については、表 2-1 を参照してください。

表 2-1. LVGL ハードウェア接続の例

LP-MSPM33C321A	TFT ディスプレイ
PA8	SPI SCLK
PA9	SPI PICO
PA10	リセット
PB3	PWM
PB6	LCD SCS
PB30	LCD SDC
PC5 (使用しません)	INT
PA0	SDA
PA1	SCL
PC4	RST

2.2 ソフトウェア

以下では、LVGL のサンプルに対して行われたソフトウェア設定について説明します:

- 構成
  - MCU:以下の値は、このサンプル向けに特別に設定されています:
    - GPIO:この例では、表 2-2 で定義されているように、複数の GPIO ピンを初期的に high または low に設定するように構成します。これらの値は、ディスプレイドライバおよびタッチ ドライバの要件に応じて異なります。

表 2-2. LVGL、GPIO ピン構成の例

ピン名	方向	初期値	割り当てられたピン
LCD_SCS	出力	設定	PB6
LCD_SDC	出力	クリア	PB30
LCD_RESET	出力	設定	PA10
LCD_PWM	出力	クリア	PB3
CTP_RST	出力	設定	PC4
CTP_INT	入力	該当なし	PC5

    - SYSCTL: SYSCTL モジュールは、外部の高周波クロック入力を MCLK の入力源として使用するように設定されています。HFXT は 40MHz で実行するように設定されています
    - I2C:選択された I2C ペリフェラルである UC3 は、コントローラとして動作し、高速モード (400kHz) で動作するように設定されており、SDA および SCL ラインはそれぞれ PA0 と PA1 に割り当てられています
    - SPI:選択された SPI ペリフェラルである UC2 は、コントローラとして動作し、40Mhz で動作するように設定されています。チップ セレクトには CS1 を使用し、Motorola 4 線式モードで動作し、SCLK、PICO、POCI、CS1 はそれぞれ PA8、PA9、PB19、PB6 に割り当てられています

    - TIMER：選択されたタイマ モジュール TIMG8_0 は、BUSCLK をクロック源とし、分周比 40 でプリスケールされ、周期 1ms の周期ダウンカウント モードで動作するように設定されています。タイマのゼロ イベントでトリガされる割り込みが設定されています
  - LVGL：
    - この例では、LVGL は使用されている TFT ディスプレイの仕様で構成されています。LVGL のデモでは多くの異なるモジュールが使用されているため、オプションのフォントやウィジェットの多くが有効化されています。LVGL のデモをコンパイルして実行できるように、デモ用設定値 LV_USE_DEMO_WIDGETS も特別に有効化されています。

```
#define MY_DISP_HOR_RES        480
#define MY_DISP_VER_RES        320
...
#define LV_COLOR_DEPTH          16
...
#define LV_MEM_SIZE   (48U * 1024U) // [bytes]
...
#define LV_DRAW_COMPLEX 1
...
#define LV_USE_DEMO_WIDGETS 1
```

- 初期化
  - 最初に初期化する必要があるのは、SysConfig を通じて事前に設定されたペリフェラルです
  - 次に、システム ティックをトリガするために使用されるタイマ割り込みを初期化する必要があります
  - ペリフェラルとタイマの後、LVGL 固有の項目を初期化する必要があります
    - LVGL 初期化：LVGL は lv_init を呼び出すことで初期化する必要があります
    - 出力初期化：LVGL のディスプレイドライバは、lv_port_disp_init を呼び出すことで初期化されます
      - このサンプルでは、lv_port_disp_init はセクション 1.3 で定義されている初期化手順に従って処理を行います。flush コールバックは、このセクションの後半で定義されています

```
void lv_port_disp_init(void)
{
    disp_init(); // Initialize your display

    static lv_disp_draw_buf_t draw_buf_dsc_1;
    static lv_color_t buf_1[MY_DISP_HOR_RES * 10];                          /*A buffer
for 10 rows*/
    lv_disp_draw_buf_init(&draw_buf_dsc_1, buf_1, NULL, MY_DISP_HOR_RES * 10);   /
*Initialize the display buffer*/

    static lv_disp_drv_t disp_drv;                          /*Descriptor of a display
driver*/
    lv_disp_drv_init(&disp_drv);                    /*Basic initialization*/

    disp_drv.hor_res = MY_DISP_HOR_RES;
    disp_drv.ver_res = MY_DISP_VER_RES;

    disp_drv.flush_cb = disp_flush;

    disp_drv.draw_buf = &draw_buf_dsc_1;

    lv_disp_drv_register(&disp_drv);
}
```

    - 入力初期化：LVGL の入力デバイスドライバは、lv_port_indev_init を呼び出すことで初期化されます
      - このサンプルでは、lv_port_indev_init は セクション 1.3 で定義されている初期化手順に従って処理を行います。入力読み取りコールバックは、このセクションの後半で定義されています

```
void lv_port_indev_init(void)
{
    static lv_indev_drv_t indev_drv;

    touchpad_init(); //Initialize your touchpad
```

```
    lv_indev_drv_init(&indev_drv); //Register a touchpad input device
    indev_drv.type = LV_INDEV_TYPE_POINTER;
    indev_drv.read_cb = touchpad_read;
    indev_touchpad = lv_indev_drv_register(&indev_drv);
}
```

- 最後に、システム ティック タイマが起動します

```
int main(void)
{
    SYSCFG_DL_init();

    NVIC_EnableIRQ(TIMER_0_INST_INT_IRQN);

    lv_init(); // init LVGL
    lv_port_disp_init();
    lv_port_indev_init();

    DL_TimerG_startCounter(TIMER_0_INST);
```

- 出力
  - LVGL は、初期化セクションで事前に定義された flush コールバックを呼び出すことで、ピクセル データを送信します。このサンプルでは、disp_flush が低レベルのドライバコードである lvgl_LCD_Color_Fill を呼び出します。この関数が SPI 通信を処理し、その後、フラッシュ処理が完了したことを LVGL に通知します

```
static void disp_flush(lv_disp_drv_t * disp_drv, const lv_area_t * area, lv_color_t * color_p)
{
    lvgl_LCD_Color_Fill(area->x1, area->y1, area->x2, area->y2, color_p);
    lv_disp_flush_ready(disp_drv); //Inform the graphics library that you are ready with the
flushing
}
```

- 入力
  - LVGL は、初期化セクションで事前に定義された入力読み取りコールバック メソッドを呼び出すことで、ユーザー入力を取得しようとします。このサンプルでは、touchpad_read が低レベルのドライバ コードを呼び出し、I2C 通信でタッチパッドが押されたかどうかを確認し、押されたことを検し出た場合は、押された位置を処理します

```
static void touchpad_read(lv_indev_drv_t * indev_drv, lv_indev_data_t * data)
{
    static lv_coord_t last_x = 0;
    static lv_coord_t last_y = 0;

    if(touchpad_is_pressed()) {
        touchpad_get_xy(&last_x, &last_y); //Save the pressed coordinates and the state
        data->state = LV_INDEV_STATE_PR;
    }
    else {
        data->state = LV_INDEV_STATE_REL;
    }

    data->point.x = last_x; //Set the last pressed coordinates
    data->point.y = last_y;
}
```

- 更新
  - LVGL のシステム ティックは、先に設定した TIMG8_0 タイマ割り込みによってトリガされます。この割り込みはミリ秒ごとにトリガされました。

```
void TIMER_0_INST_IRQHandler(void)
{
    switch (DL_TimerG_getPendingInterrupt(TIMER_0_INST)) {
        case DL_TIMER_IIDX_ZERO:

            lv_tick_inc(1);     //LVGL Heart Beat

            break;
        default:
```

```
            break;
    }
}
```

  - LVGL のタイマ トリガは、このサンプルのメインの while ループ内で処理されています。CPU の処理を占有しすぎないように、320 サイクルの遅延が設けられています。

```
while (1) {
        // update LVGL task
        lv_task_handler();

        delay_cycles(320);
    }
```

- メイン デモ
  - メイン デモは、LVGL ライブラリに含まれている lv_demo_widgets に定義されています。このデモでは、LVGL に含まれている主なウィジェットを紹介します。このデモを使用するため、このデモのメイン ソース ファイルにデモ用のヘッダ ファイルを含めました。

```
#include "lv_demo_widgets.h"
...
int main(void)
{
...
    lv_demo_widgets();
...
}
```

2.3 LVGL のまとめ例

LP-MSPM33C321A と TFT ディスプレイ間のハードウェア接続が完了し、ソフトウェアの変更をコンパイルして M33 デバイスに書き込むと、M33 と TFT ディスプレイが相互に通信し、LVGL デモが実行されます。このデモでは、ボタン、チャート、画像など、複数のウィジェットを表示するディスプレイを操作できます。

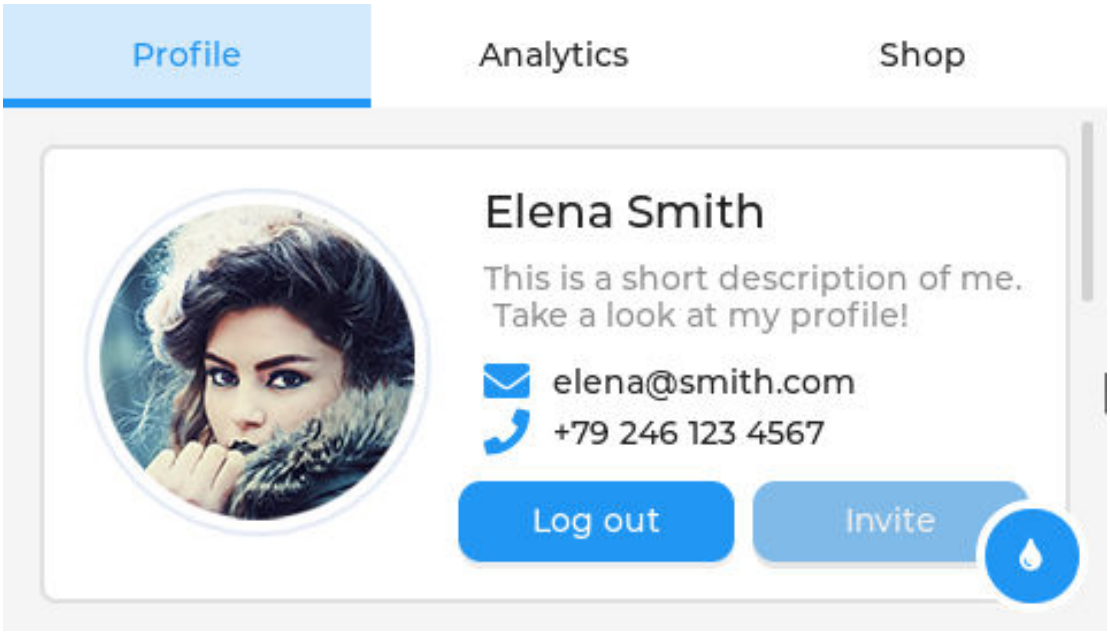

図 2-1. LVGL デモ ユーザー インターフェイス

3 まとめ

このドキュメントは、LVGL の概要と、それを MSPM33 の組み込み GUI に適切に統合するために考慮すべき点について説明するところから始まります。その後、このアプリケーション ノートでは、SPI ディスプレイドライバと I2C タッチパッド ドライバを使用して、LVGL の基本デモを再現する LVGL の使用例について説明しています。

4 改訂履歴

資料番号末尾の英字は改訂を表しています。その改訂履歴は英語版に準じています。

日付	改訂	注
2025 年 12 月	*	初版リリース

重要なお知らせと免責事項

TI は、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、TI 製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した TI 製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている TI 製品を使用するアプリケーションの開発の目的でのみ、TI はその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。TI や第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、 TI の販売条件 、 TI の総合的な品質ガイドライン 、 ti.com または TI 製品などに関連して提供される他の適用条件に従い提供されます。TI がこれらのリソースを提供することは、適用される TI の保証または他の保証の放棄の拡大や変更を意味するものではありません。 TI がカスタム、またはカスタマー仕様として明示的に指定していない限り、TI の製品は標準的なカタログに掲載される汎用機器です。

お客様がいかなる追加条項または代替条項を提案する場合も、TI はそれらに異議を唱え、拒否します。

最終更新日：2025 年 10 月