# Harmony Graphics Training using SAM9X60 Curiosity

# Table of Contents

# Hardware Pre-requisite

Here is the list of hardware required to complete this lab:

- EV40E67A SAM9X60 Curiosity Development Board

  https://www.microchip.com/en-us/development-tool/EV40E67A

- A Micro-USB cable to power the board
- USB to TTL Serial 3.3V UART Converter Cable (Optional)

  https://www.amazon.com/FTDI-TTL-232R-3V3-SERIAL-CONVERTER-CABLE/dp/B00DDF8TV6/ref=asc_df_B00DDF8TV6/?tag=hyprod-20&linkCode=df0&hvadid=242167970588&hvpos=&hvnetw=g&hvrand=2616582347583421449&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=1014034&hvtargid=pla-571447074621&psc=1

  Link to driver for the USB cable:

  https://ftdichip.com/drivers/vcp-drivers/

- High-Performance WVGA Display Module with Max-Touch®

  https://www.microchip.com/en-us/development-tool/AC320005-5

- External Segger JTAG debugger such as J32 Debug Probe

# Software Pre-requisite

**We will run this lab on a Windows PC**.

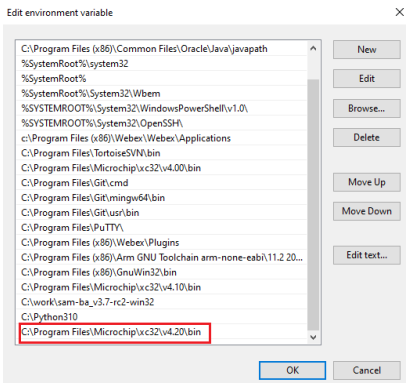
The following SW needs to be installed:

**(The following training was developed using the versions of SW mentioned below. Therefore, we strongly recommend using the same versions to avoid any un-explainable errors/behaviors.)**

- MPLAB X IDE Version 6.05
  https://ww1.microchip.com/downloads/aemDocuments/documents/DEV/ProductDocuments/SoftwareTools/MPLABX-v6.05-windows-installer.exe
  NOTE: The MPLAB X IDE version 6.05 comes with the MCC plugin installed.

- XC32 Compiler Suite Version 4.20
  https://ww1.microchip.com/downloads/aemDocuments/documents/DEV/ProductDocuments/SoftwareTools/xc32-v4.20-full-install-windows-x64-installer.exe

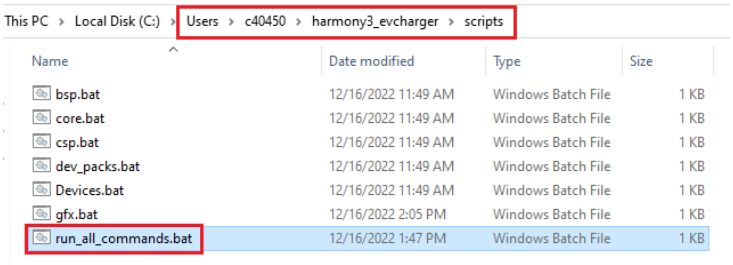
After installing the compiler, please ensure that toolchain path is included in the system environment variable:




- MPLAB Harmony 3 Framework
  The following mandatory Harmony Software Framework needs to be installed:

| Harmony Software | Version | Download Link (Github) |
|---|---|---|
| BSP | v3.14.0 | https://github.com/Microchip-MPLAB-Harmony/bsp.git |
| Core | v3.11.1 | https://github.com/Microchip-MPLAB-Harmony/core.git |
| CSP | v3.13.1 | https://github.com/Microchip-MPLAB-Harmony/csp.git |
| Devices | N/A | https://github.com/Microchip-MPLAB-Harmony/devices.git |
| Dev_Packs | v3.13.1 | https://github.com/Microchip-MPLAB-Harmony/dev_packs.git |
| GFX | v3.12.0 | https://github.com/Microchip-MPLAB-Harmony/gfx.git |

- Please use git to install the software framework. Downloading the code as a ZIP file may lead to missing files and therefore errors during generating code.
- Create a suitable folder in your work folder, for example, harmony3_evcharger in C:\Users\<ID>. Copy the “scripts” folder provided in the “resources” folder and run the “run_all_commands.bat” script to install the required SW framework listed in the table above.



The following icons are used in this lab manual:

The expected result of a task

Pay special attention

For your information

Technical information about a specific topic

Additional task

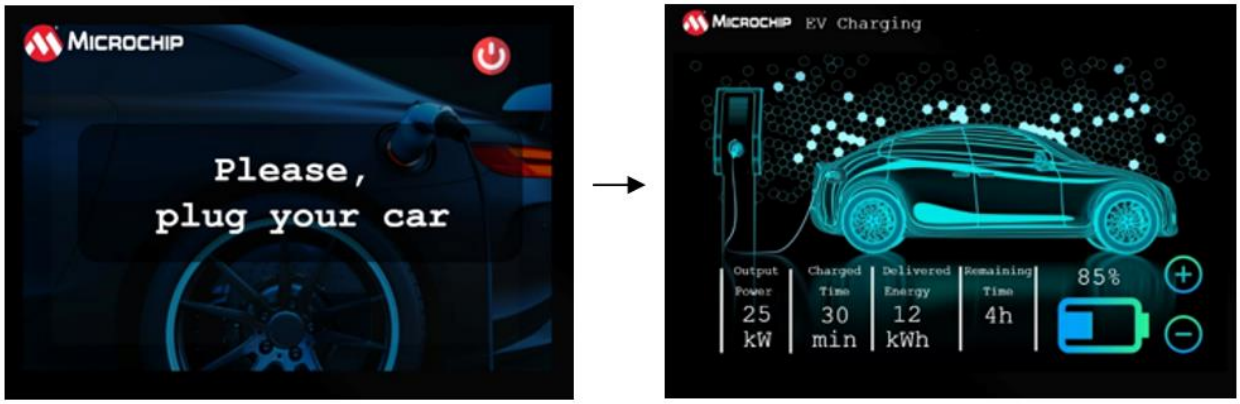
# INTRODUCTION

In this lab we will learn how to use MCC (MPLAB Code Configurator) and Legato Graphics Composer to develop graphical application for ARM MPU32 target boards. The SAM9X60 Curiosity Development will be used for this training.

The following topics will be covered in the lab:

- How to set-up the hardware
- Managing buttons, labels, image, image sequence widgets and Schemes
- Concept of multi layers in graphics design
- Button events, Screen events and Screen Transitions
- Dynamic Strings
- Animation using Canvas
- Use of RTC and Timers in managing graphics components

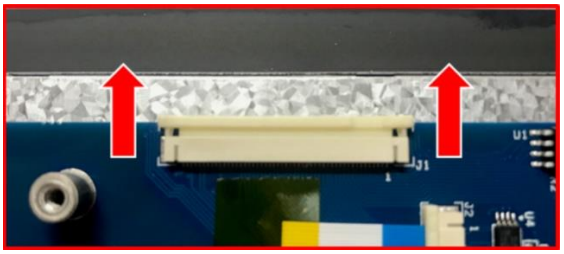
We will be working towards developing a simple EV charging graphical application as shown below:

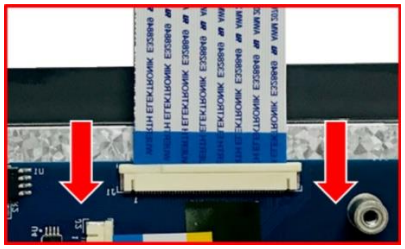



# HARDWARE SETUP

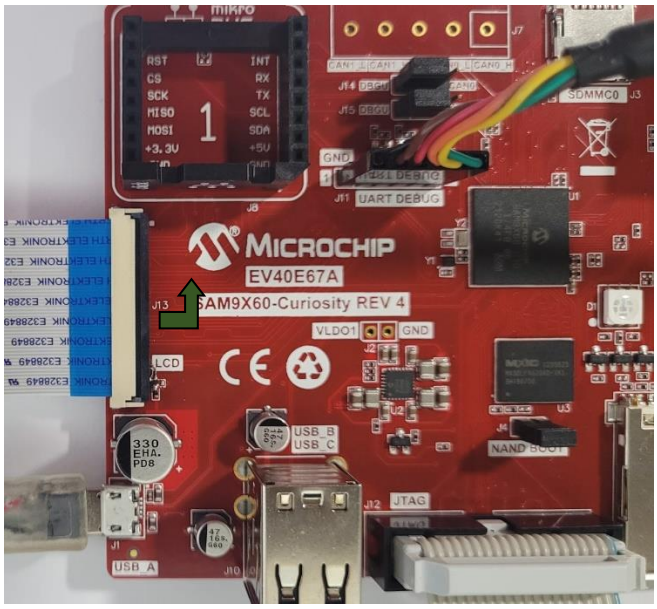
<u>Assembling the WVGA Display</u>

1. If you are using a brand new WVGA display out of the box, remove the short flexible cable from the back of the display.
2. To open the cream-colored connector, pull the latch evenly using the catch feature on each side. Insert the long flexible cable to the back of the WVGA Display ensuring the silver connectors are down and blue side is up. Close the connector by sliding the latch back.





3. On J13 LCD connector of the SAM9X60 Curiosity, open the connector by lifting the black flap. Make sure to lift it from the center so that pressure is applied evenly. Insert the cable gently, blue side up, making sure it's in all the way and close the connector by pressing the black flap down to secure the cable.

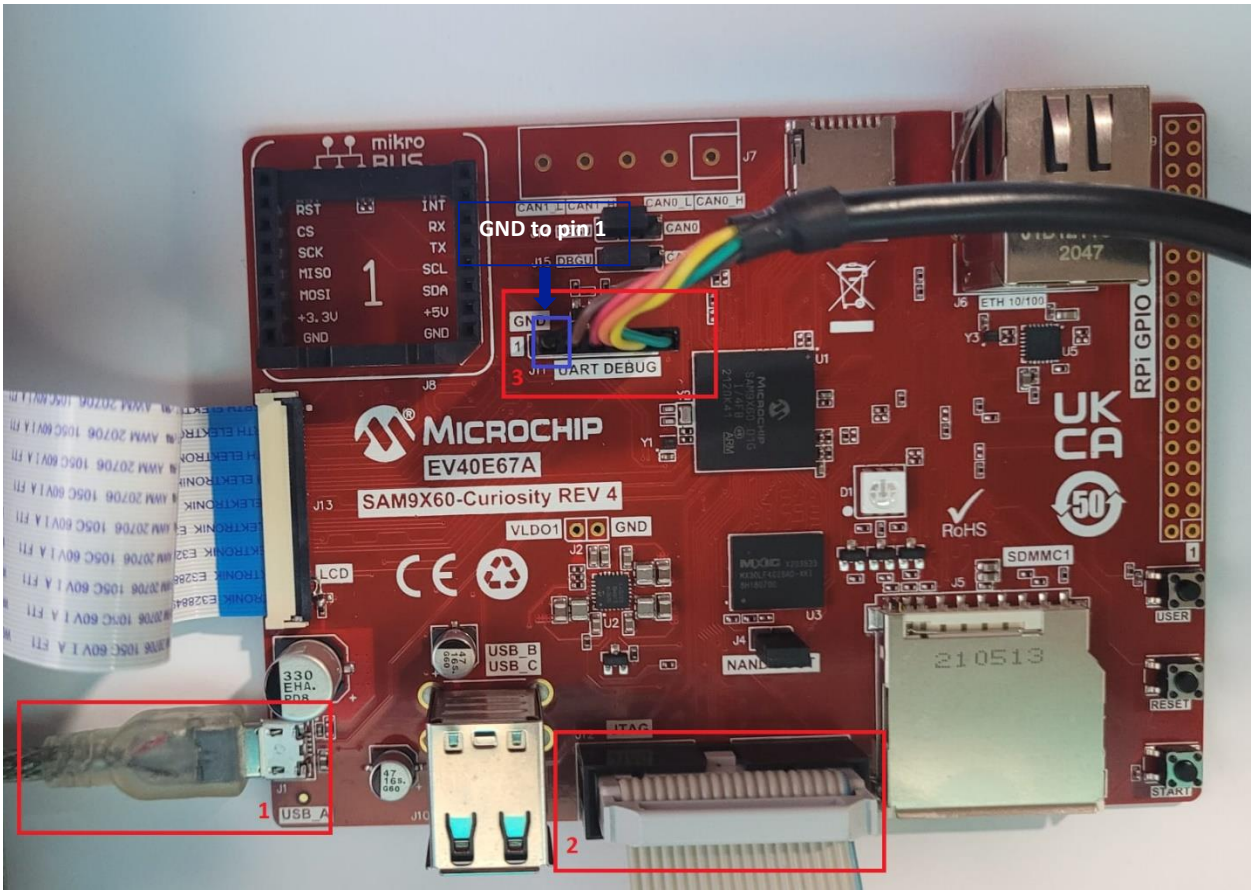



Host-Target Setup

1. Power the board by connecting a micro-USB cable to J1.
2. Connect a JTAG Debugger to J12 to allow the host to debug the target.
3. Connect the FTDI USB to UART converter cable to J11 and open a serial console using the following settings:

***Please ensure that the Black GND wire is connected to pin 1 (GND) of J11 of the target board.***

| Baud rate | 115200 |
|---|---|
| Data | 8 |
| Parity | None |
| Stop bit | 1 bit |
| Flow control | None |
| **115200 8-N-1** | |



# Task 1: Managing simple widgets and Schemes

## *Purpose:*

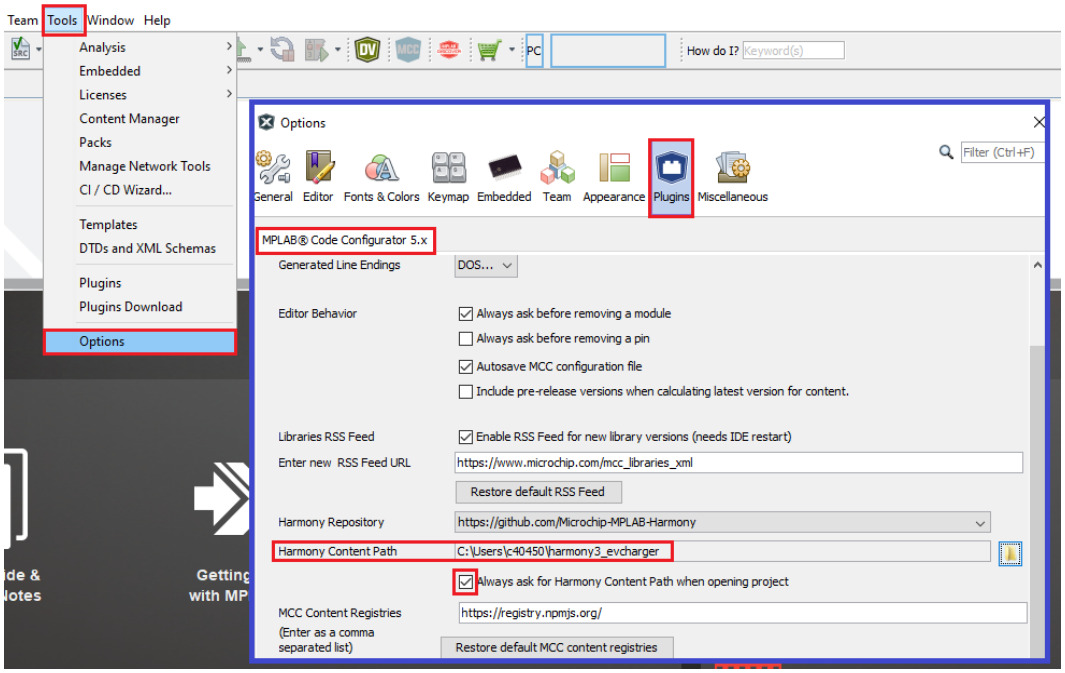
To learn how to handle simple widgets such as Image, Buttons and Labels and to manage schemes.

## *Overview:*

In this task we create an MCC project from scratch and using Legato Graphics Composer, we will create a simple GUI with button, image and label widgets. We will learn how to add a new font, string, and modify schemes. We will compile and debug the project using MPLAB X IDE on the SAM9X60 Curiosity development board.

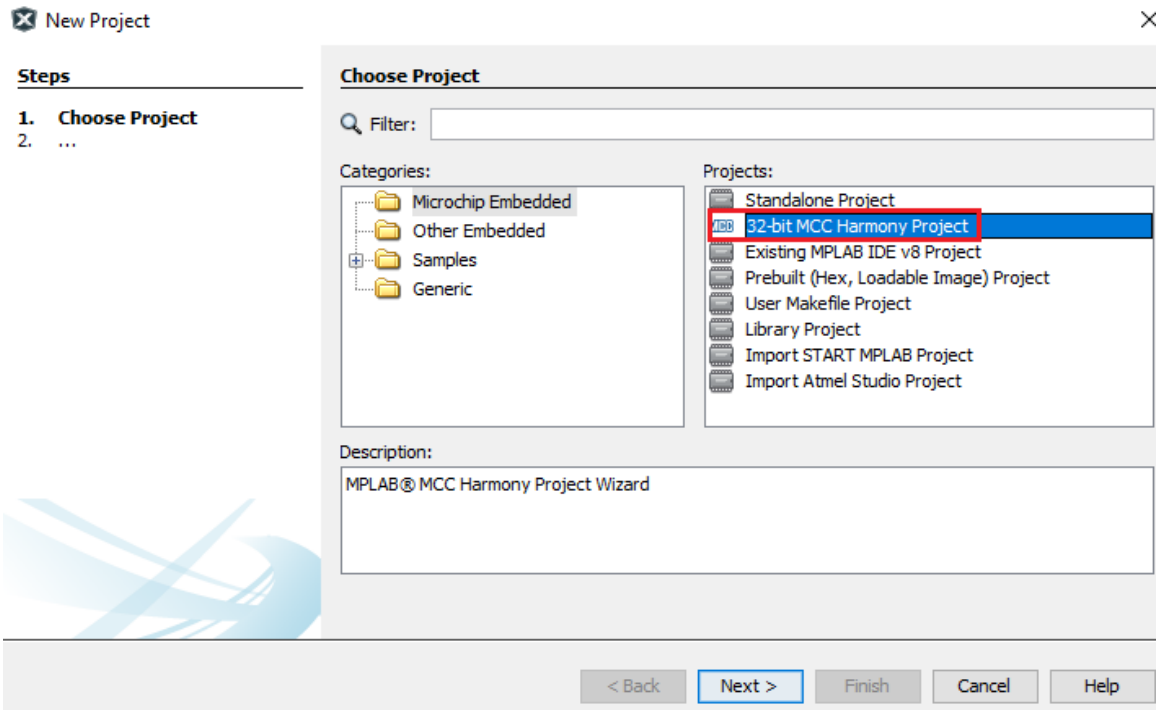
## *Procedure:*

***Before you start this task, click on "Tools" from the main menu and select "Options". From the resulting window, select "Plugins" and select "MPLAB Code Configurator" tab. Ensure that the "Harmony Content Path" is populated and is pointing to the folder where you downloaded the Harmony code. You may optionally choose to select the "Always ask for Harmony Content Path when opening project".***

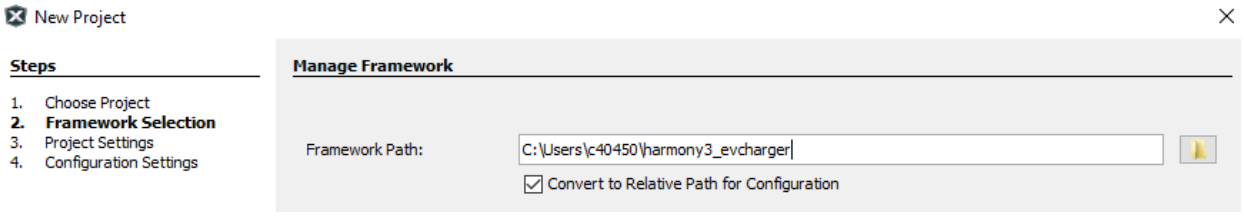



<u>Creating New Harmony Project</u>

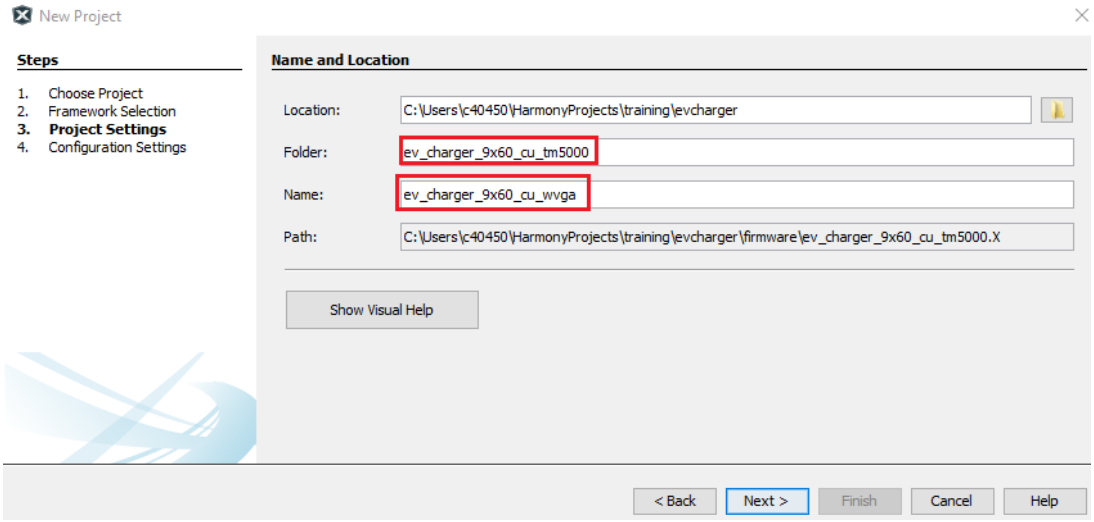
1. Click on File -> New Project. Please make sure you choose "32-bit CC Harmony Project":



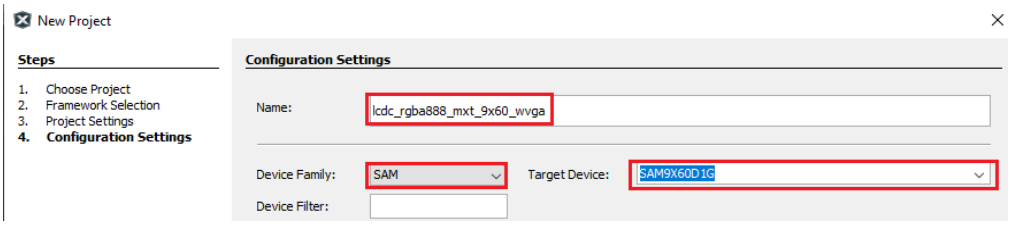

2. For Framework Path, choose the location where Harmony repository is downloaded – "Core, CSP, GFX, etc):



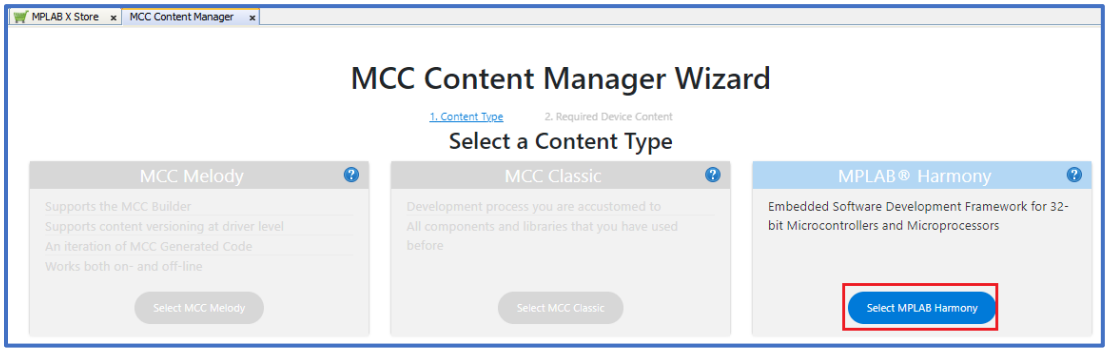

3. Enter the Project location, name and folder name as shown below:
   *Project Location:* <path>/evcharger
   *Project Folder:* evcharger_9x60_cu_tm5000
   *Project Name:* evcharger_9x60_cu_wvga

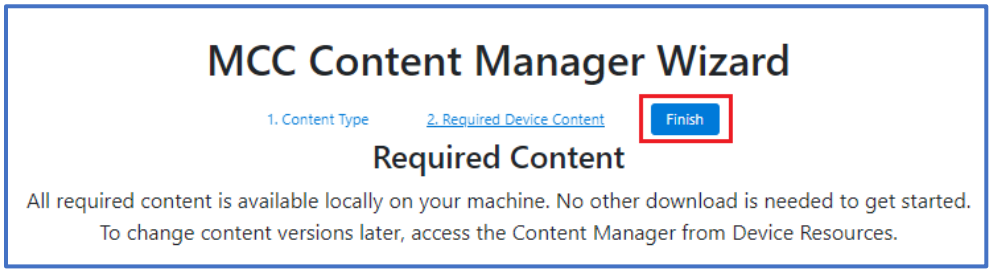


4. Select device and choose a suitable configuration name and select Finish:
   *Configuration Name:* lcdc_rgba888_mxt_9x60_wvga



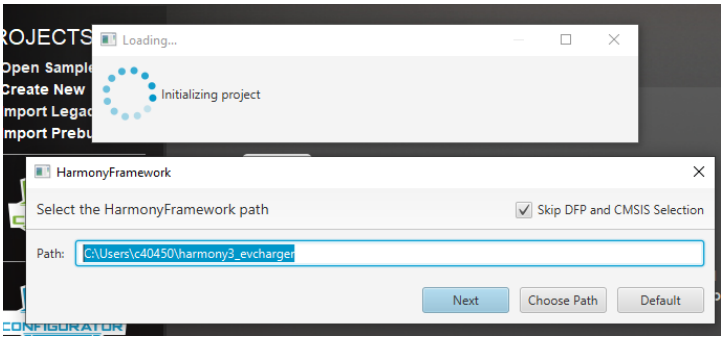

***At this point MCC is automatically launched. Click on "Select MPLAB Harmony" and click "Finish".***




***MCC can be launched at any point by clicking on the following icon:*** 

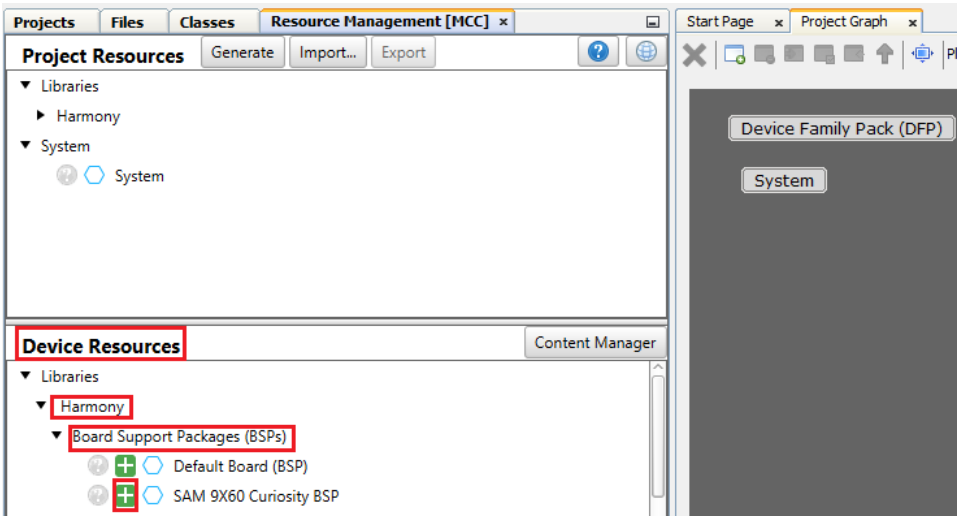

5. **If** you see the following window, click "Next":



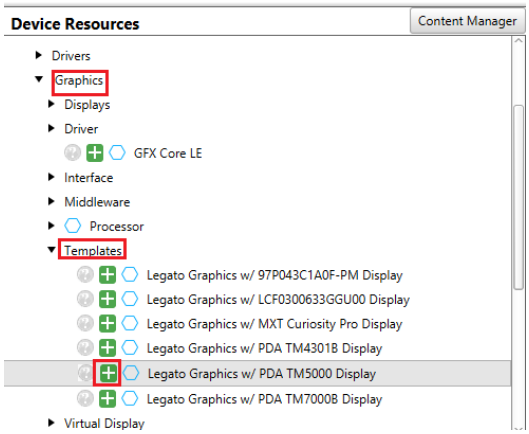

<u>Configuring Project Graph using MCC</u>

6. From the Device Resources Window select the following drivers:

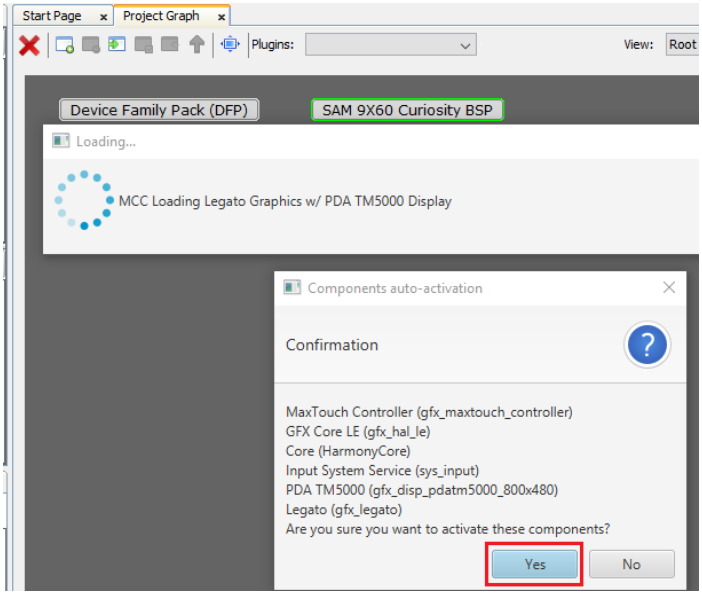
i. Harmony -> Board Support Packages (BSPs) -> Click on the green + sign next to “SAM 9X60 Curiosity BSP”.




ii. Harmony -> Graphics -> Templates -> Click on the green + sign next to “Legato Graphics w/ PDA TM5000 Display”

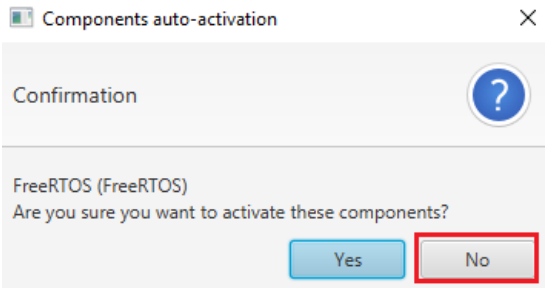


You will be shown a “Component auto-activation” confirmation box. Please select “Yes”.



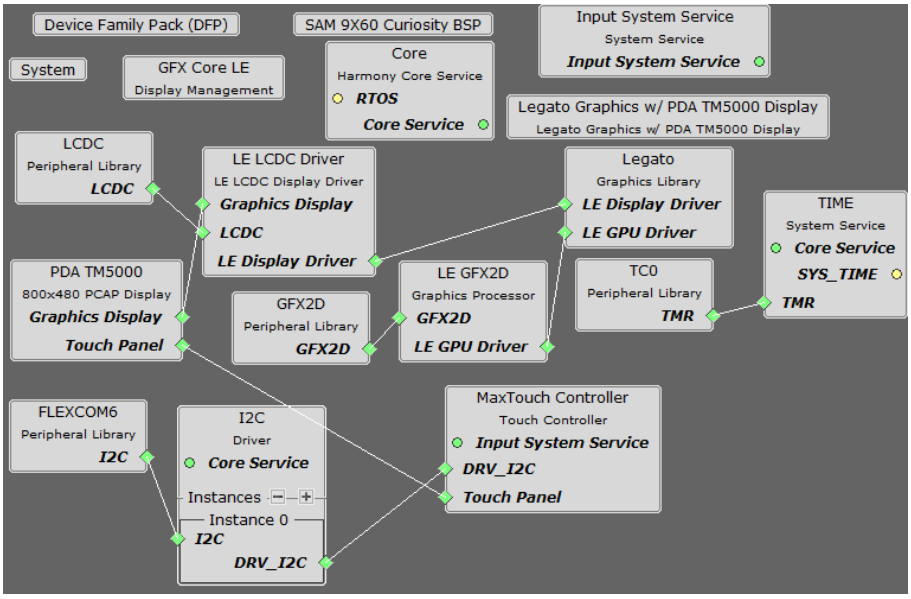

‘

**Please select “No” for FreeRTOS:**

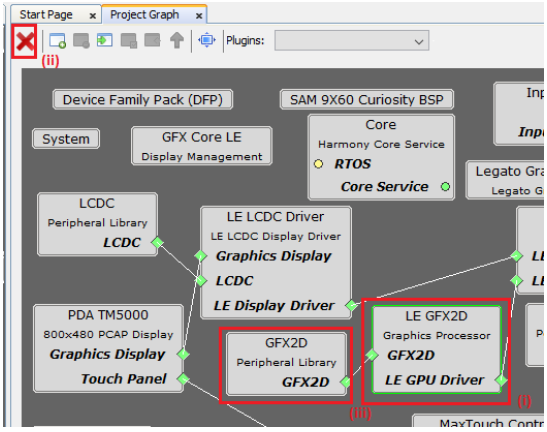



**In subsequent confirmation boxes, please select “Yes”.**

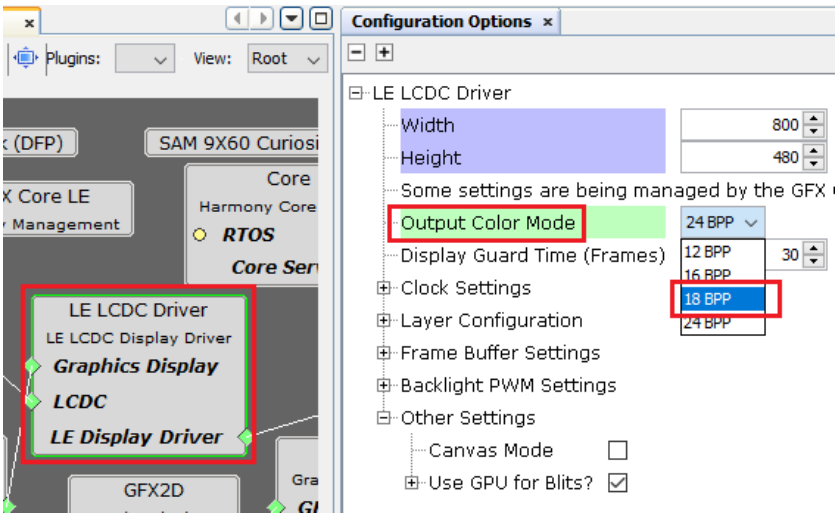
You should see a project graph generated as below:



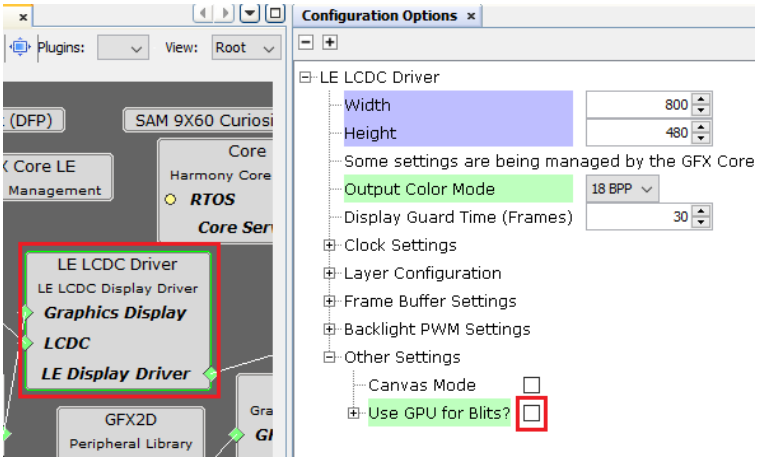

7. At this point, we will remove the 2D GPU from the project graph.
   (i) Select “LE GFX2D” component
   (ii) Select the red cross on the top left of the Project Graph window to remove the component
   (iii) Similarly remove the “GFX2D” component.




8. From the LCD pin schematics shown above, we can see that only 18 pins are used for LCD data. Therefore select “LE LCDC Driver” component from the Project graph and in the “Configuration Options”, for “Output Color Mode”, choose 18BPP.

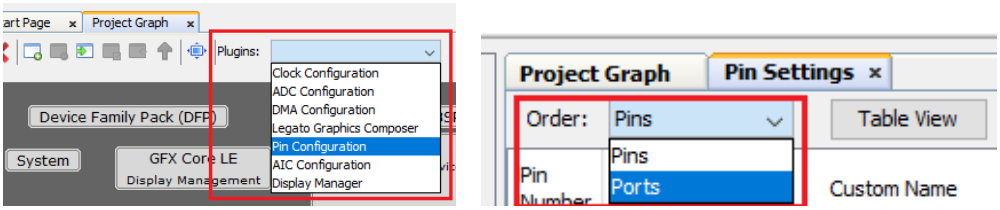



9. With the “LE LCDC Driver” component selected, in the “Configuration Options”, uncheck the box for “Use GPU for Blits”.




<u>Pin Configuration</u>

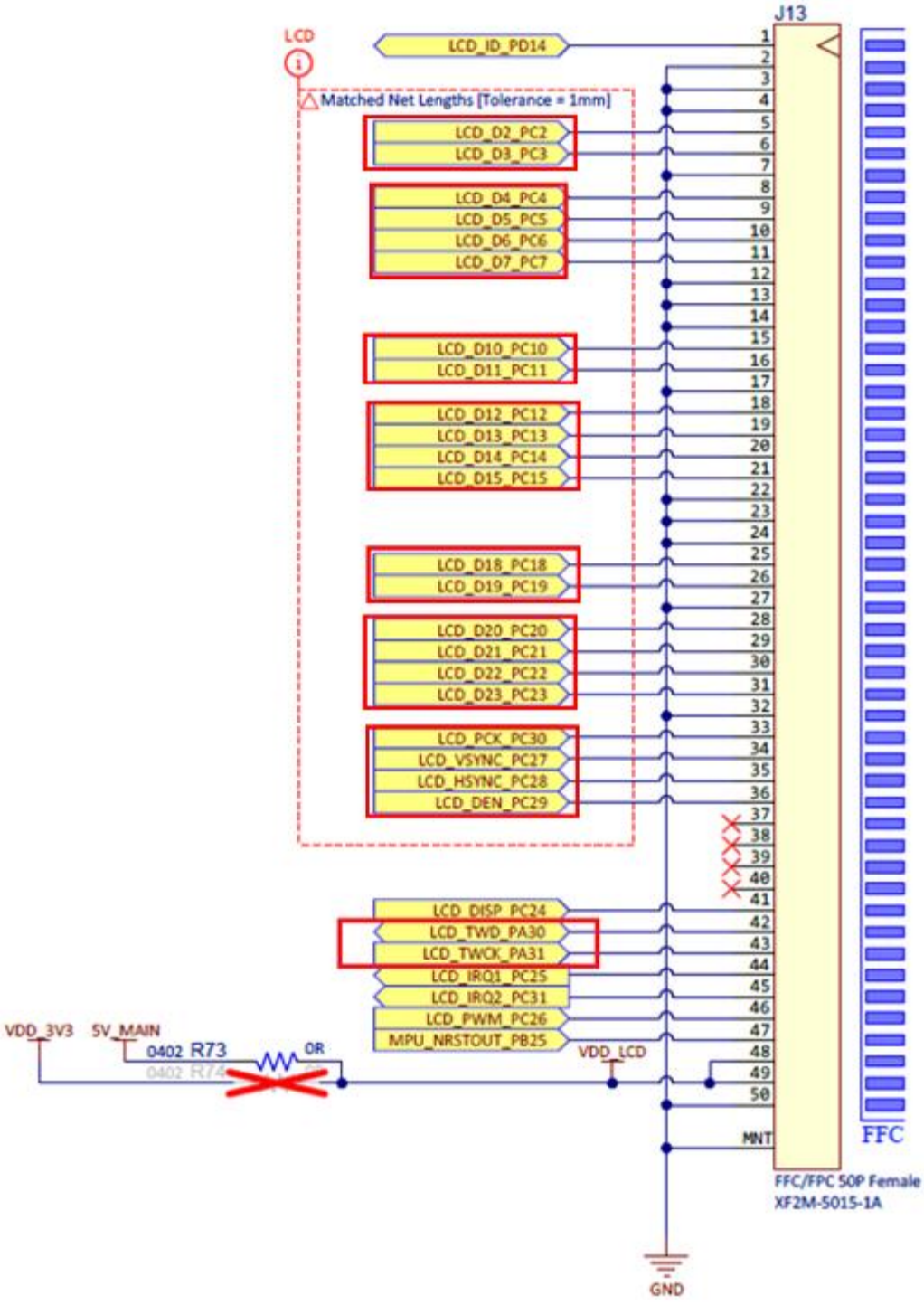
10. From the Plugins drop down menu, choose “Pin Configuration”. In the Pin Settings tab, from the “Order” drop down menu, choose “Ports”.




We configure the LCD pins based on the LCD pins schematics shown in the page below. Set the pins to the configuration as shown below:

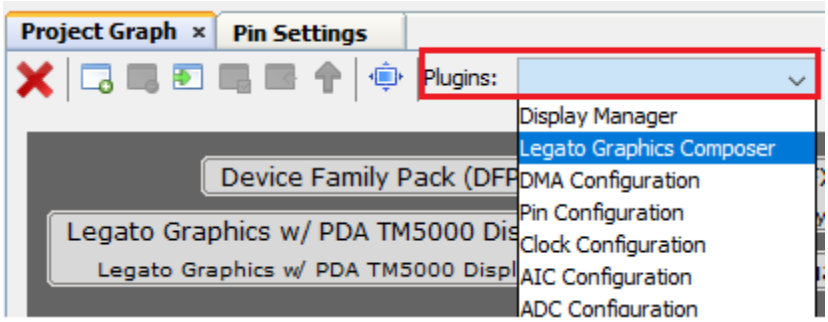
| Pin Number | Pin ID | Custom Name | Function | Direction |
|---|---|---|---|---|
| B7 | PB25 | | Available | In |
| U8 | PC0 | | Available | In |
| U3 | PC1 | | Available | In |
| T4 | PC2 | | LCDC_LCDDAT2 | n/a |
| T1 | PC3 | | LCDC_LCDDAT3 | n/a |
| R5 | PC4 | | LCDC_LCDDAT4 | n/a |
| U4 | PC5 | | LCDC_LCDDAT5 | n/a |
| T7 | PC6 | | LCDC_LCDDAT6 | n/a |
| R2 | PC7 | | LCDC_LCDDAT7 | n/a |
| U7 | PC8 | | Available | In |
| R1 | PC9 | | Available | In |
| T8 | PC10 | | LCDC_LCDDAT10 | n/a |
| T3 | PC11 | | LCDC_LCDDAT11 | n/a |
| T5 | PC12 | | LCDC_LCDDAT12 | n/a |
| R4 | PC13 | | LCDC_LCDDAT13 | n/a |
| U2 | PC14 | | LCDC_LCDDAT14 | n/a |
| U5 | PC15 | | LCDC_LCDDAT15 | n/a |
| R10 | PC16 | | Available | In |
| R8 | PC17 | | Available | In |
| R7 | PC18 | | LCDC_LCDDAT18 | n/a |
| P6 | PC19 | | LCDC_LCDDAT19 | n/a |
| T14 | PC20 | | LCDC_LCDDAT20 | n/a |
| P8 | PC21 | | LCDC_LCDDAT21 | n/a |
| R14 | PC22 | | LCDC_LCDDAT22 | n/a |
| T9 | PC23 | | LCDC_LCDDAT23 | n/a |
| U14 | PC24 | | LCDC_LCDDISP | n/a |
| P5 | PC25 | BSP_MAXTOUCH_CHG | GPIO | In |
| R13 | PC26 | | LCDC_LCDPWM | n/a |
| U9 | PC27 | | LCDC_LCDVSYNC | n/a |
| P13 | PC28 | | LCDC_LCDHSYNC | n/a |
| P9 | PC29 | | LCDC_LCDDEN | n/a |
| T13 | PC30 | | LCDC_LCDPCK | n/a |

## LCD Connector

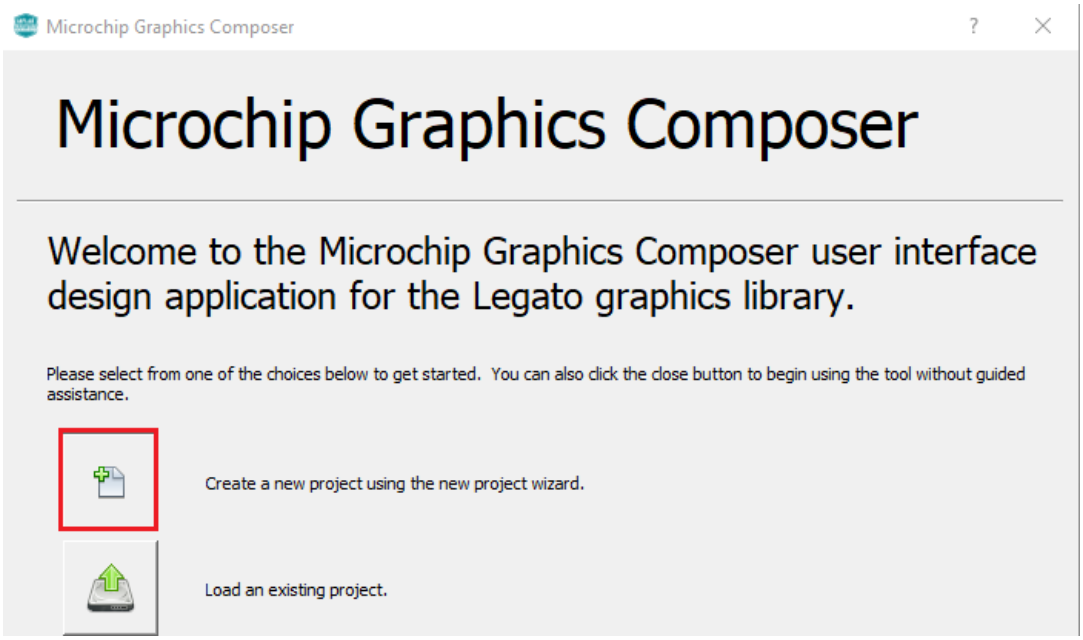


Legato Graphics Composer

11. From Project Graph Tab, select “Legato Graphics Composer” from the Plugins drop down menu:

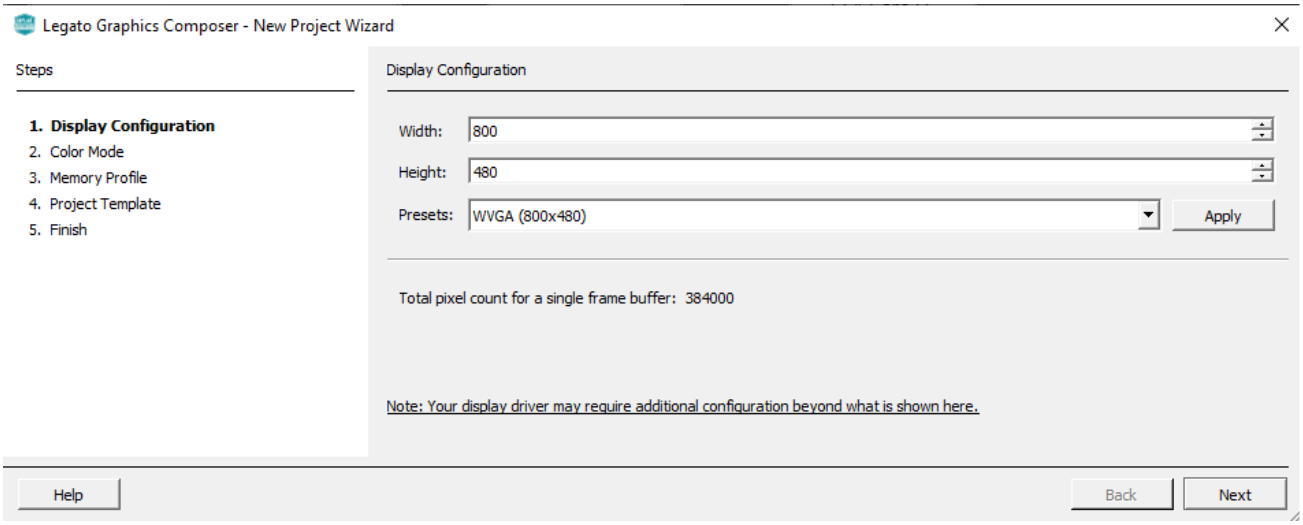



This will launch Microchip Graphics Composer. Select the “Create a new project using the new project wizard icon”.

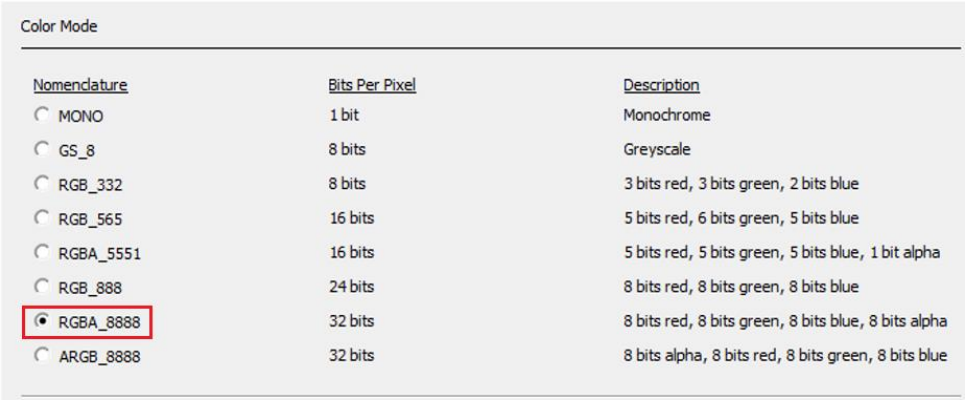



Now we will create a project template using Microchip Graphics Composer wizard:

i. Enter Width, Height and Presets information as shown below and click on Next:

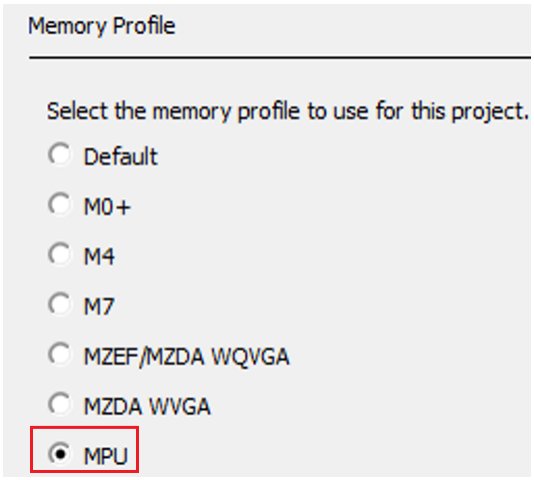



ii. Select “RGBA_8888” for color mode and click on Next:

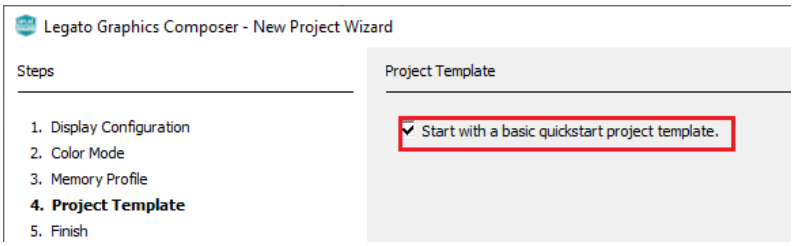



***As you can see in the description, RGBA_8888 stands for 8 bits for each of the colors – red, blue & green and 8 bits for the alpha channel (alpha controls the degree of transparency of a color)***

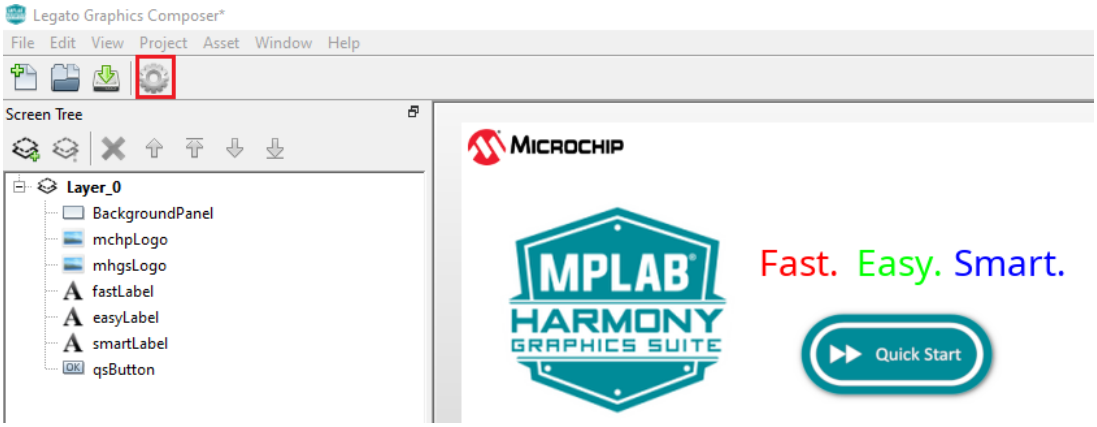
iii. For Memory Profile select MPU:




iv. Select the “Start with a basic quickstart project template” and click Next and Finish.



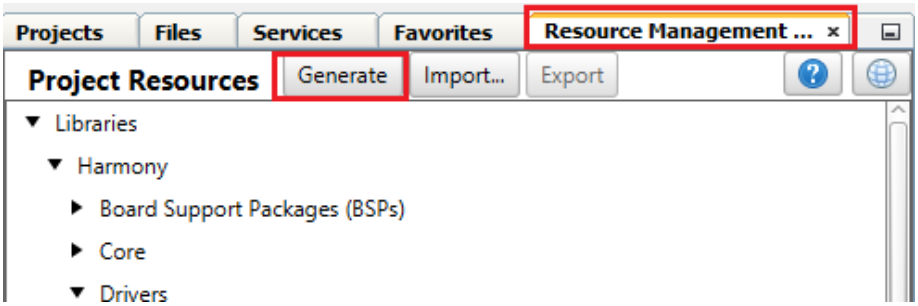

v. This generates a basic template project. Click on Generate code icon:




This generates the design files.

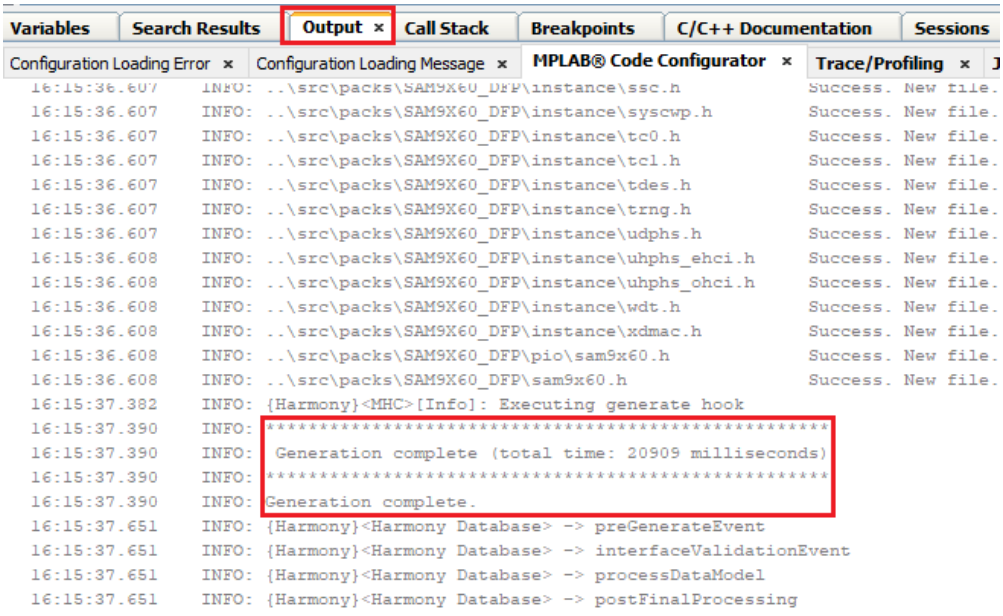
Generating Project

12. With this, all the configuration is complete. Going back to the MPLABX IDE, in the MCC Project Resources window (in the Resource Management tab), click on the “Generate” button:

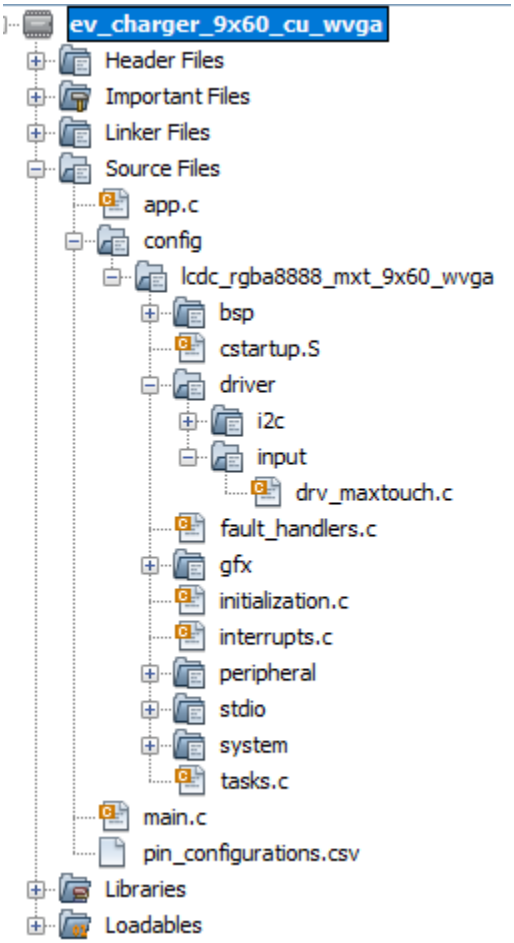



***The first time you generate a project, it will take several minutes to complete, depending on the speed of your computer. Please be patient.***

Once this process is complete, you should see the MPLAB Code Configurator log as shown below:




Now you should see the project source files generated as shown below:




13. In evcharger\firmware\src\config\lcdc_rgba888_mxt_9x60_wvga\gfx\legato\renderer\legato_renderer.c, modify the following lines:

```
#ifndef LE_NO_CACHE_ATTR
#define LE_NO_CACHE_ATTR
#endif
```

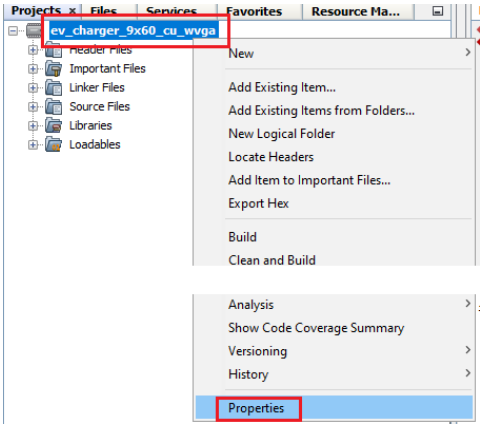
to:

```
#ifndef LE_NO_CACHE_ATTR
// CUSTOM CODE - DO NOT OVERWRITE
#define LE_NO_CACHE_ATTR SECTION(".region_nocache")
// END CUSTOM CODE
#endif
```

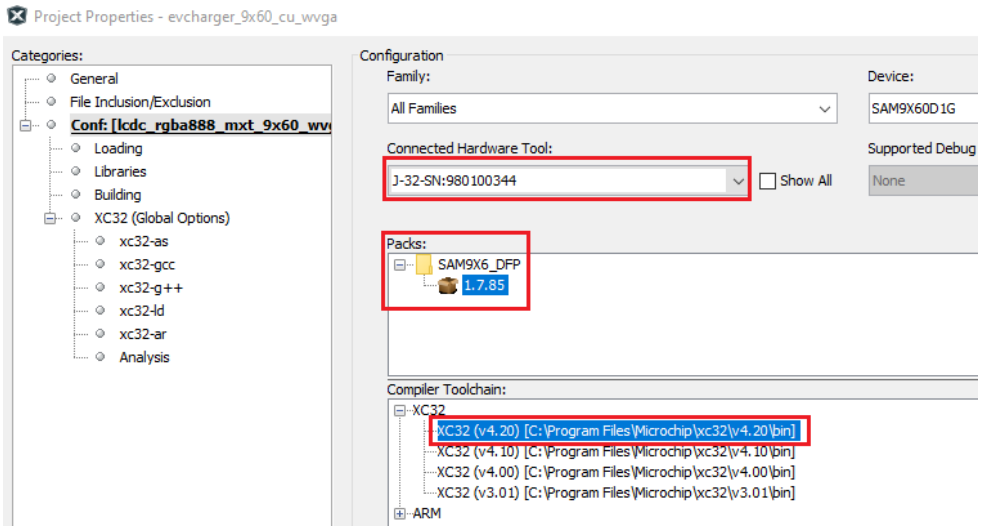
Setting Project Properties

***Before we set the project properties, please make sure that the target board is connected to the power and debugger. Also connect an FTDI UART/USB cable.***

14. Right click on the project and select “Project Properties”.

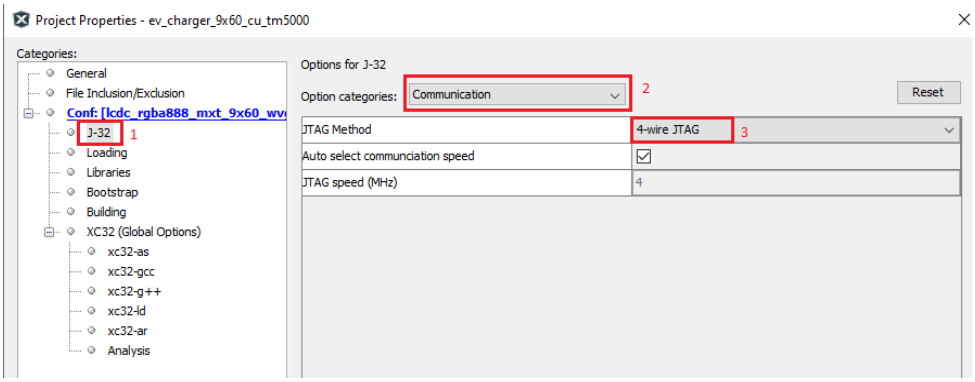



15. For “Connected Hardware Tool”, choose the debugger in use. For Packs, choose the latest version (1.7.85) and for XC32 compiler, choose the latest version (4.20) and click Apply.

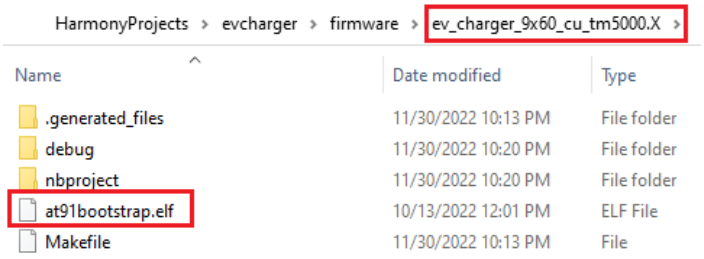



***In the snapshot above you see J-32 because the J32 Probe was used.***

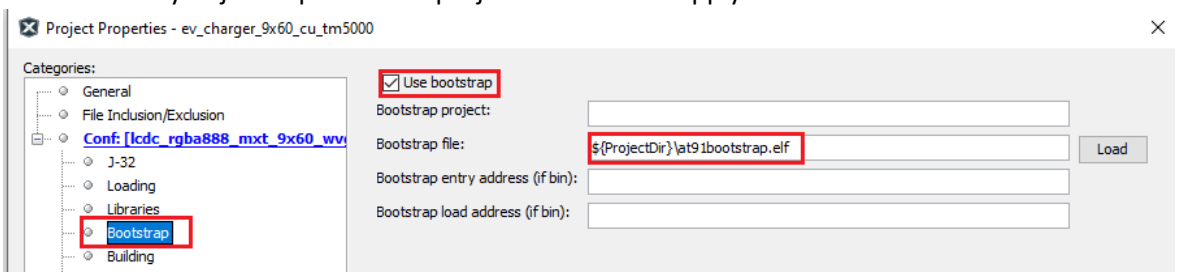
16. Click on J-32 and under “Option Categories” choose “Communication” and for JTAG Method, choose “4-wire JTAG”. Click Apply.



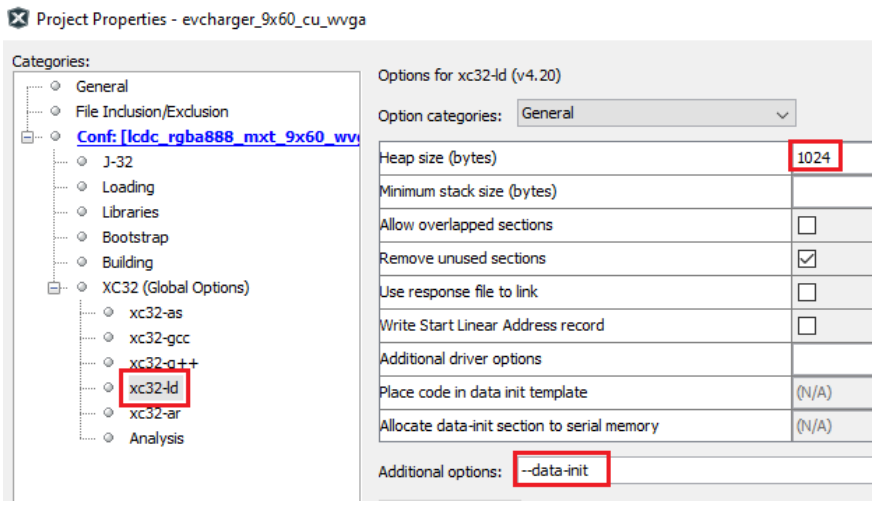

17. In the training “resources” folder, you will find “at91bootstrap.elf”. Copy this to the project folder as shown below.




Click on “Bootstrap” category and select the “Use bootstrap: checkbox. For the “Bootstrap file”, select the file you just copied to the project folder. Click Apply.




18. Click on xc32-ld category and do the following:
    - Increase heap size to ‘1024’
    - For “Additional options”, enter: --data-init
    - Click Apply



Click on Apply and OK.

Changes to startup Script and Linker file

19. With MPLAB X IDE 6.05, there is a bug that causes data abort when "`setbuf(stdin, NULL);`" is called because data memory is not initialized. To fix this
    - Add the following to the cstartup.s:

```
/* Initialize the C library */
        bl      __pic32c_data_initialization
        bl      __libc_init_array
```

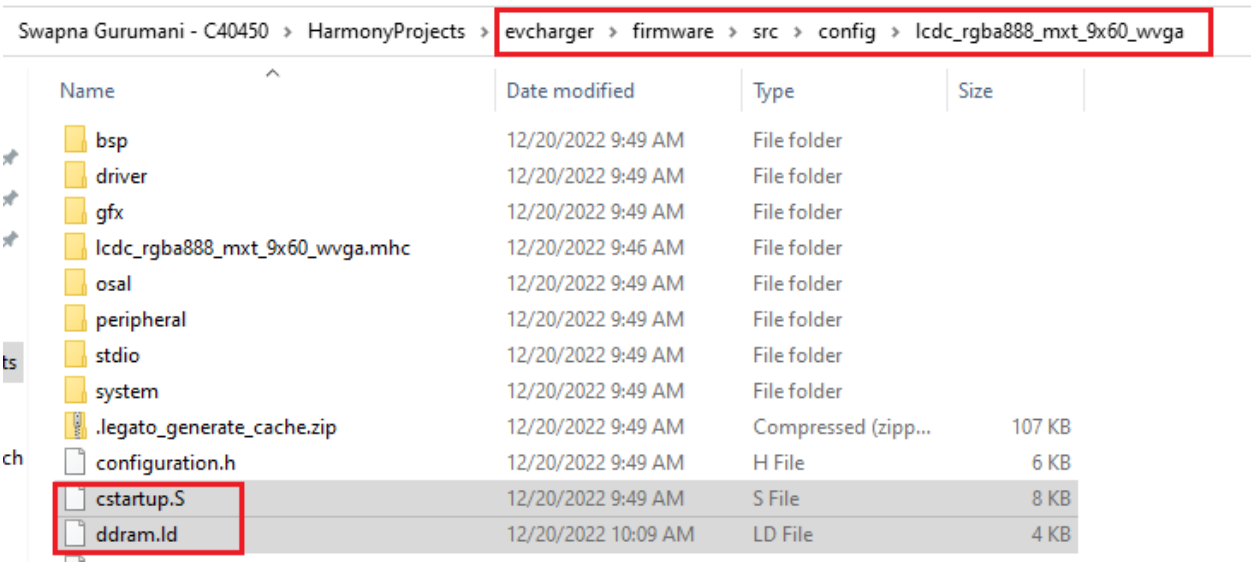
    - In the ddram.ld, modify the following lines:

```
    *(.dinit*)
                . = ALIGN(4);
        } >ram

        .data :
    {
                *(.data .data.*);
                . = ALIGN(4);
                PROVIDE(_efixed = .);             /* End of text section */
    } >ram
```

    to:

```
    //*(.dinit*)
                . = ALIGN(4);
        } >ram
    .dinit :
    {
         *(.dinit*)
        . = ALIGN(4);
    } >ram
        .data :
    {
                *(.data .data.*);
                . = ALIGN(4);
                PROVIDE(_efixed = .);             /* End of text section */
    } >ram
```

***In the training "resources" folder, you will find modified "ddram.ld" and "cstartup.S". Feel free to copy these files into your project folder at the location: evcharger\firmware\src\config\lcdc_rgba888_mxt_9x60_wvga***



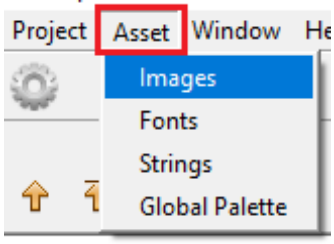

***At this point if you click on the "Debug Project" icon , the following is displayed on the LCD panel:***


## Managing Images, Fonts, Strings, and Schemes

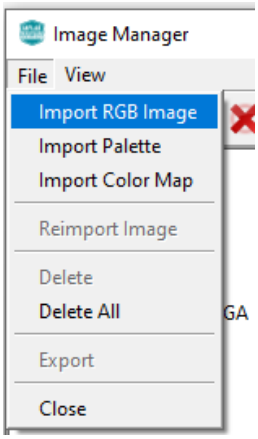
So far, we have learned how to create a Harmony MCC project from scratch and using graphics composer wizard how to create a design project. In this section we will learn how to add images, fonts and strings using Asset Manager.

**To add Images:**

i. Click on "Asset" from the Composer menu and select "Images":



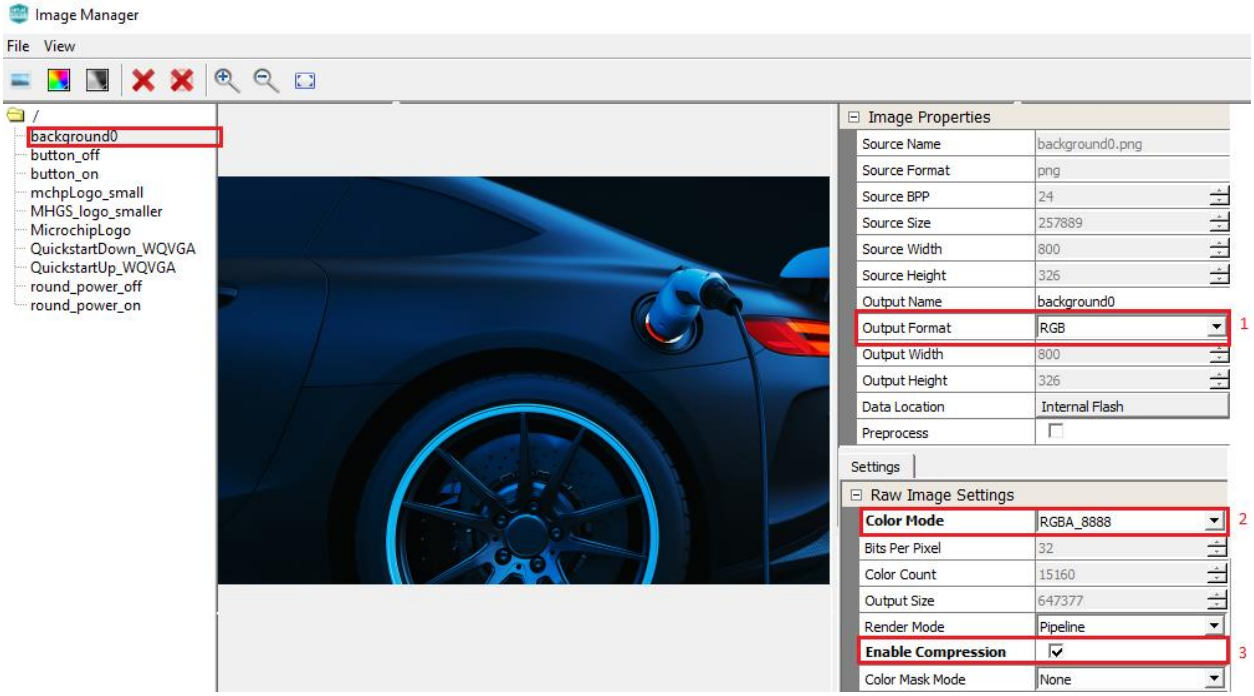

ii. Click on File -> "Import RGB Image":




In the training "resources" folder, you will find all the images to be used for this application in the "assets" folder. These images are in the correct pixel resolution.

iii. Choose the following images:
background0.png, "button_off.png", "button_on.png", "MicrochipLogo.bmp, round_power_on.png, round_power_off.png.

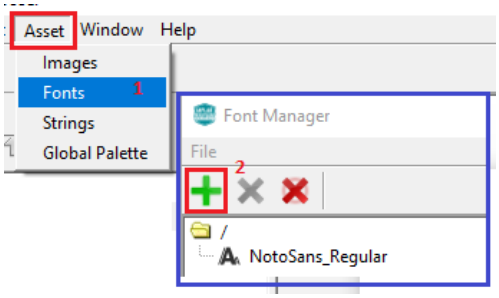
iv. Change the "Output Format", "Color Mode" and "Enable Compression" as shown below for all 6 images you just added:



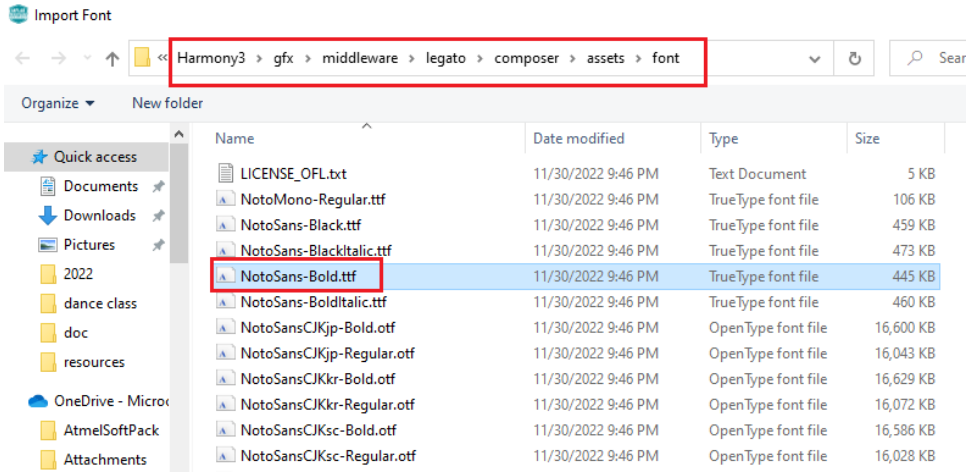

***Please make sure you make the above changes for all 6 images.***

**To add Fonts:**

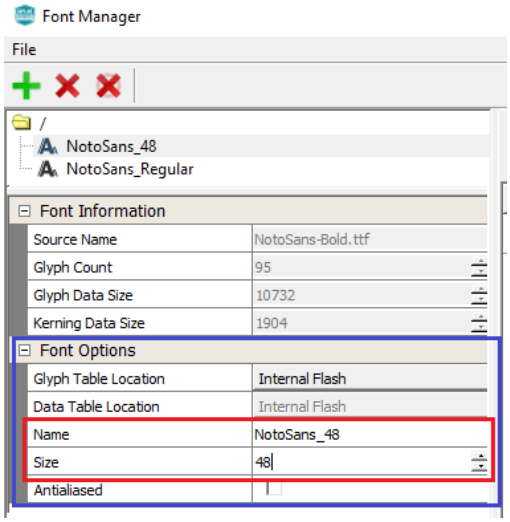
i. Click on “Asset” from the Composer menu and select “Fonts”. From the “Font Manger” window, click the + “Import Font” icon.




ii. From the browse window, select NotoSans-Bold.ttf from the location – “<path to harmony code>\harmony3_evcharger\gfx\middleware\legato\composer\assets\font”.

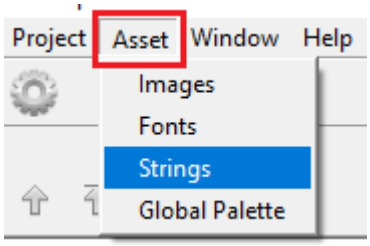



iii. Set the font name to “NotoSans48” and size to 48 from “Font Options” as shown below:

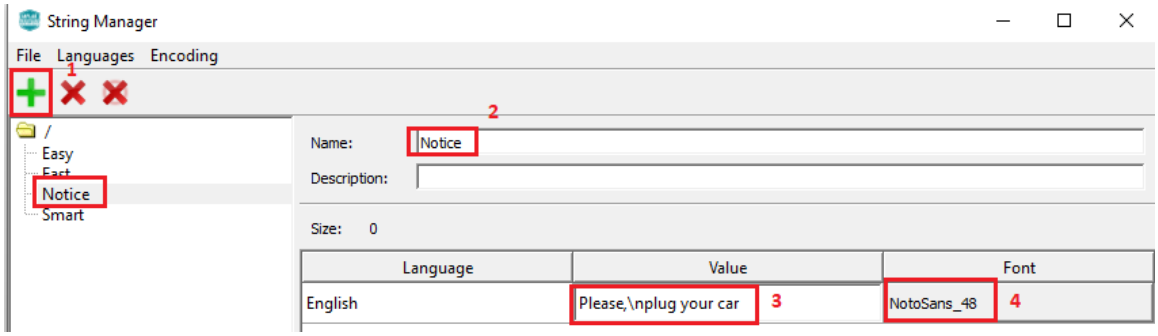


**To add Strings:**

i. Click on “Asset” from the Composer menu and select “Strings”:



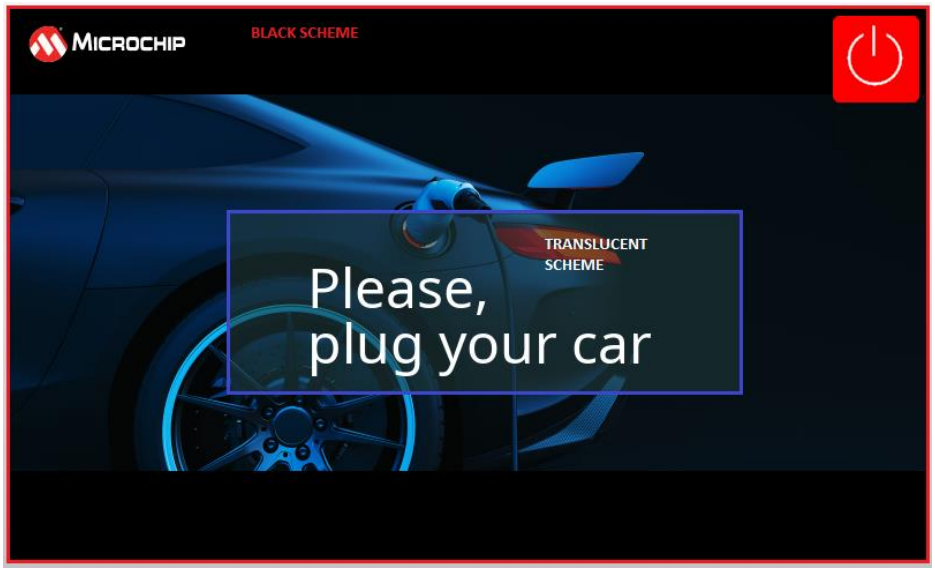

ii. Click on the green + sign to add a string, then enter “Name”, “Value” and “Font” as shown below:



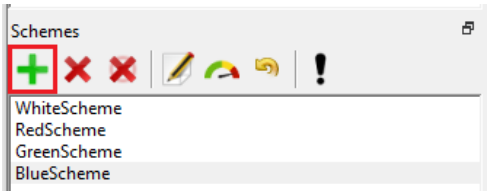

This adds a string called “Notice” with the font size 48 you created in the step above.

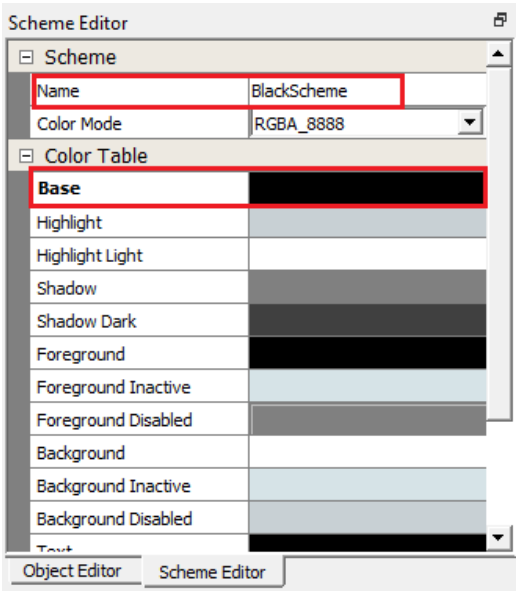
**To add Scheme:**

Schemes are used to manage look and feel of widgets by modifying color and transparency of various aspects of the widget. In the default design generated by MGC, white, red, blue, and green schemes are included. Let us add black and translucent schemes to create the black and transparent panel as shown below:

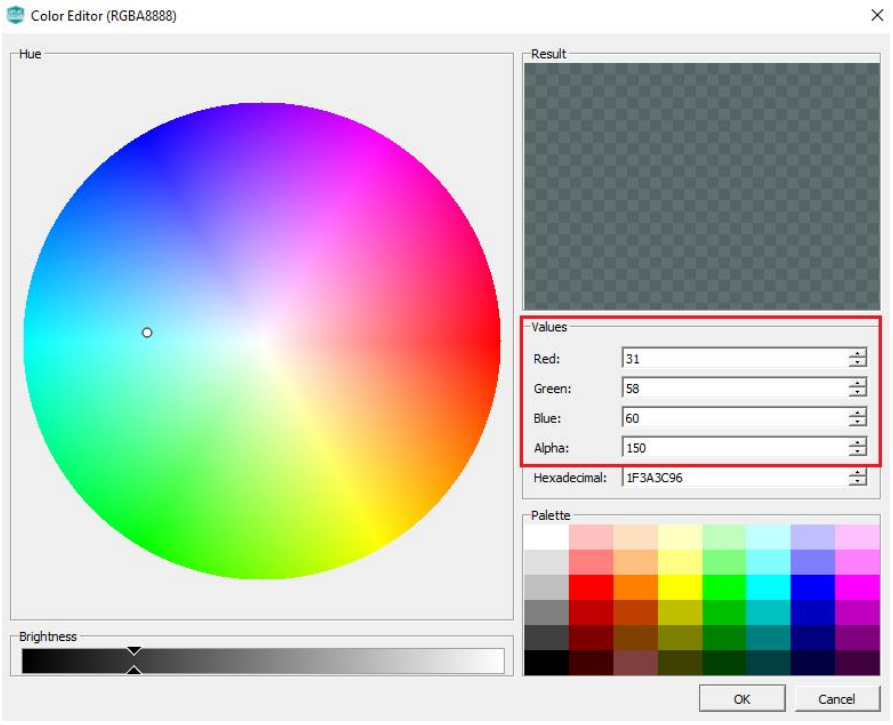


i. Click on the green + sign in the Schemes Manager to add a scheme, then in the scheme editor, enter “BlackScheme” for “Name” and click on the “Base” color and from the color editor choose black as shown below:



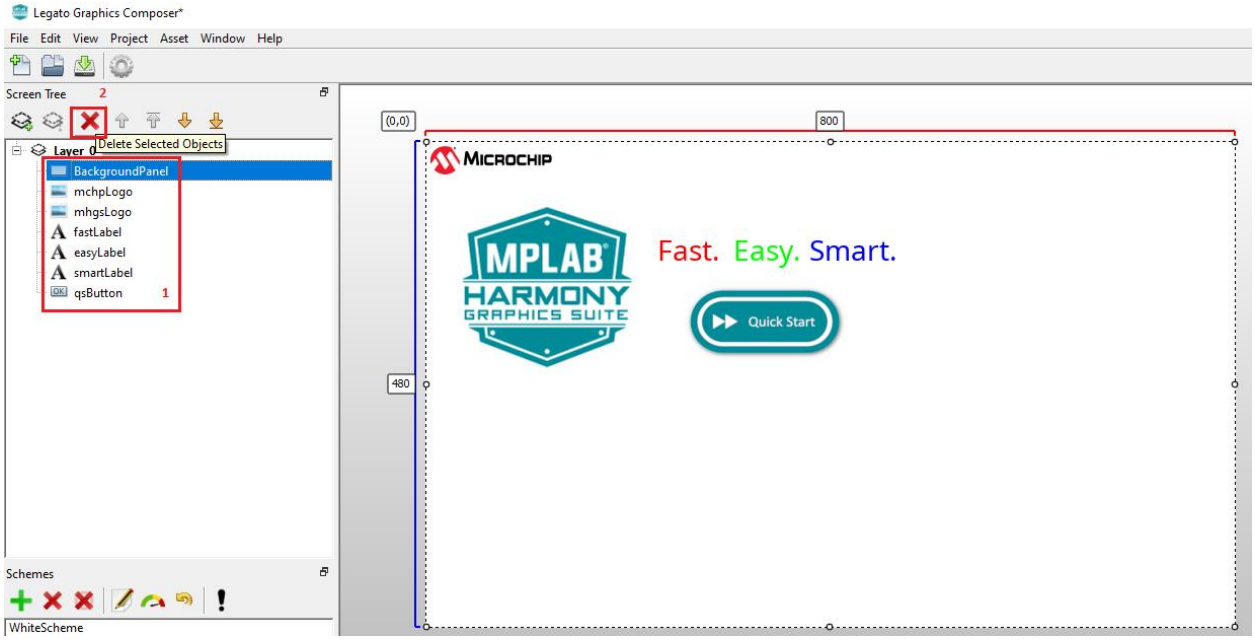

***Add a trasnparent scheme using the color editor values as shown below:***



## Designing the screen

We are now ready to design our first screen.

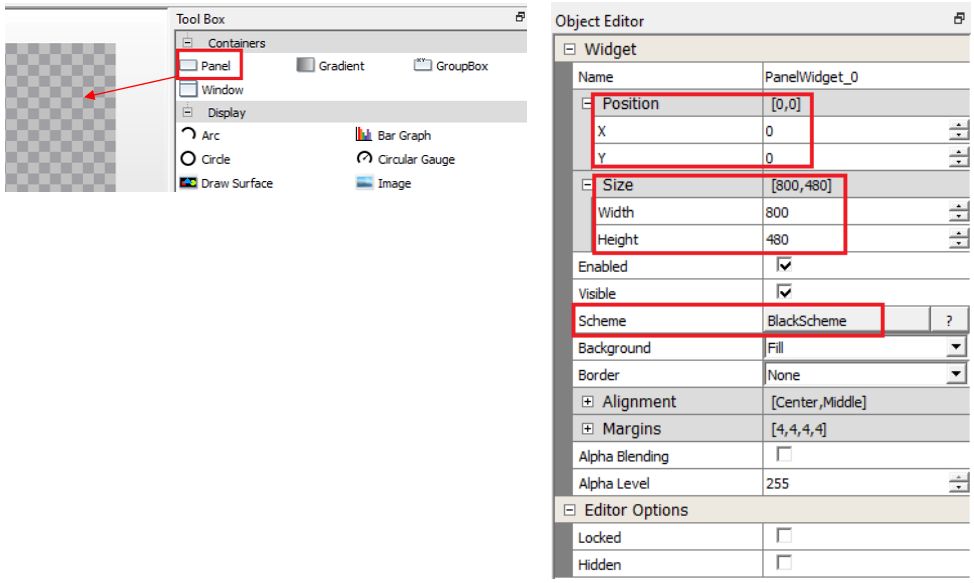
i. From the screen tree, select all the widgets and remove them by clicking on the “Delete Selected Objects” icon as shown below:




You will be asked to confirm if you want to delete selected object, select yes. Now you have a blank screen.

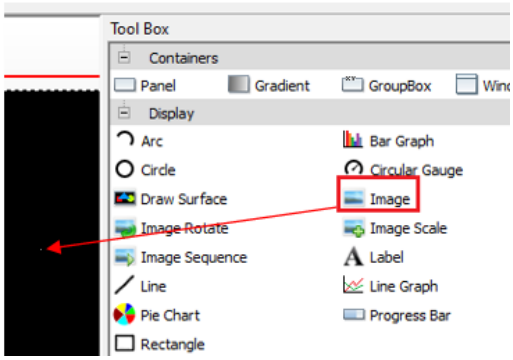
**To add a Panel:**

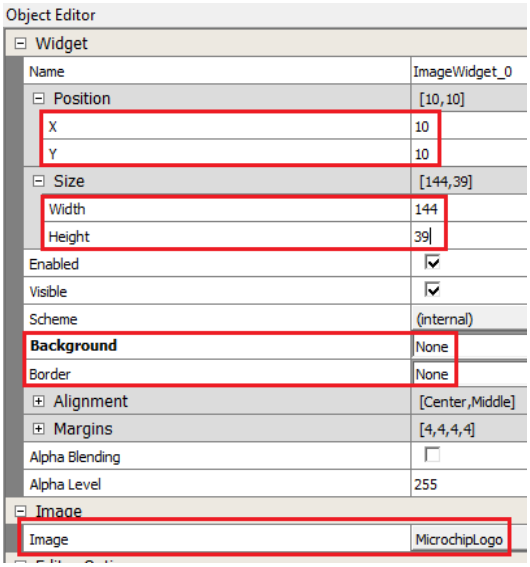
ii. From the tool box, drag the Panel widget to the design window and modify its property using object editor:



**To add Image:**

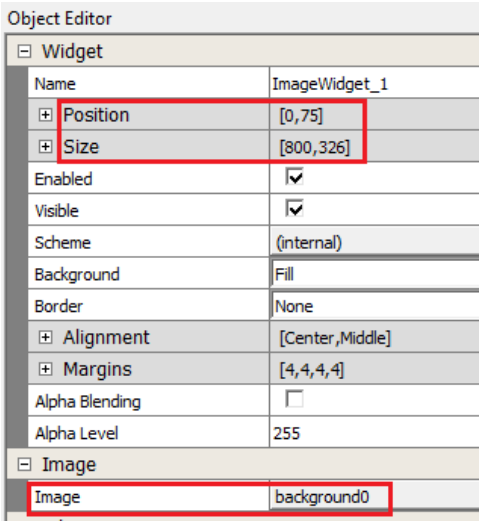
iii. From the tool box, drag the image widget to the design window and modify its property using object editor:



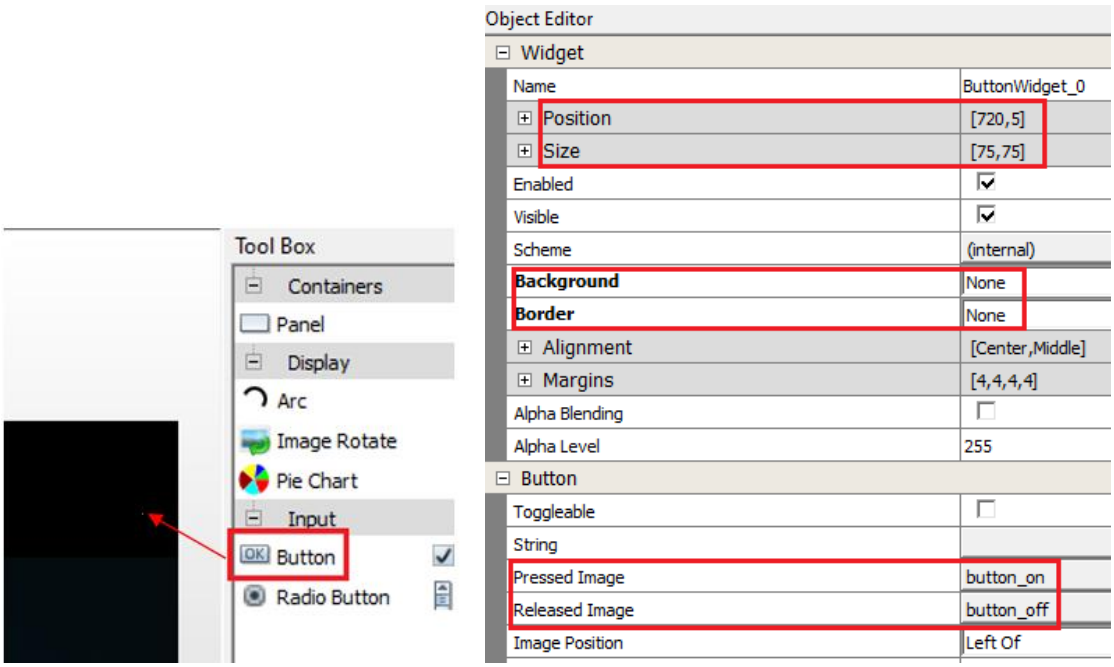

***Why is the Size of the image widget set to 144x39 (What is the unit of this measurement?)***

***Add the background image to the design using the object editor values as shown below:***



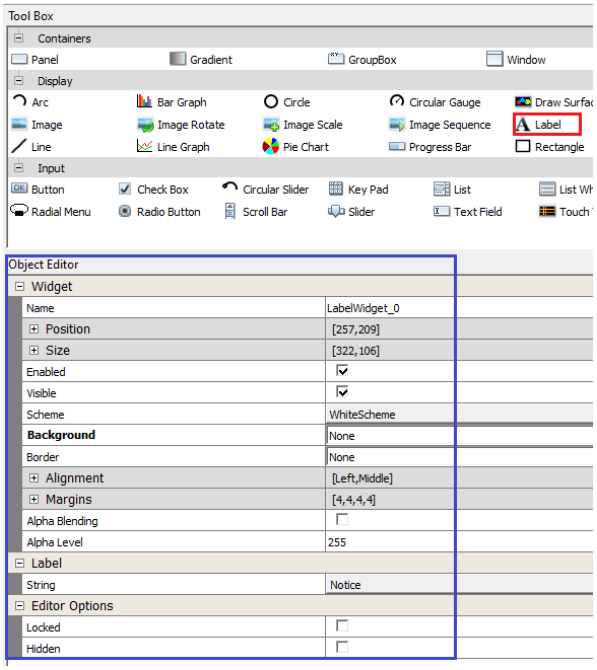
**To add User Button:**

iv. From the tool box, drag the Button widget to the design window and modify its property using object editor:




**To add Label:**

i. From the tool box, drag the Label widget to the design window and modify its property using object editor:

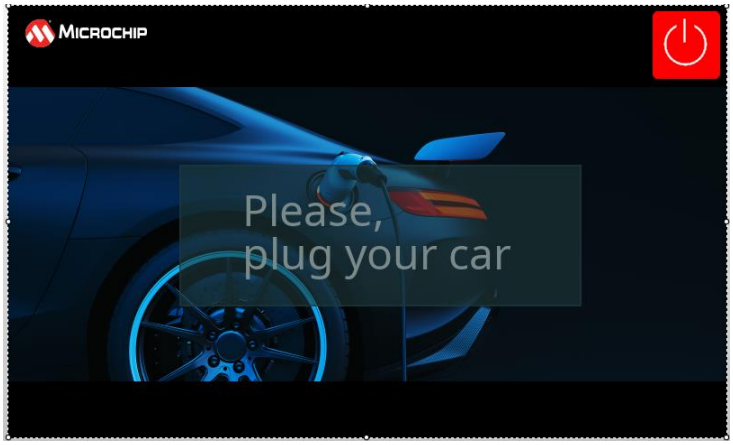


***The label you added has black font. How do you make it to look white?***

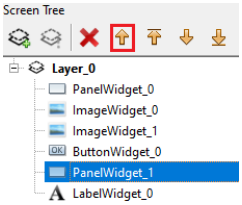
***HINT: Change the text color of the white scheme to white.***

**Bonus Task**

1. Add a rectangle with transparency behind the “LabelWidget_0”:

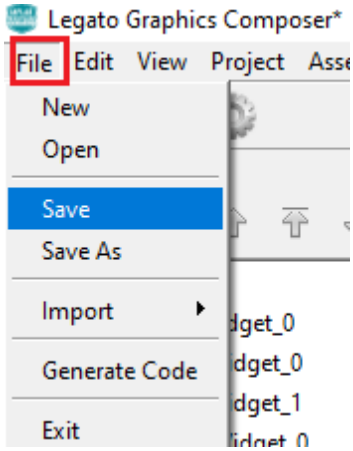



HINT: Add a rectangle panel with translucentScheme and move it behind the label using “Move selected objects up” icon:

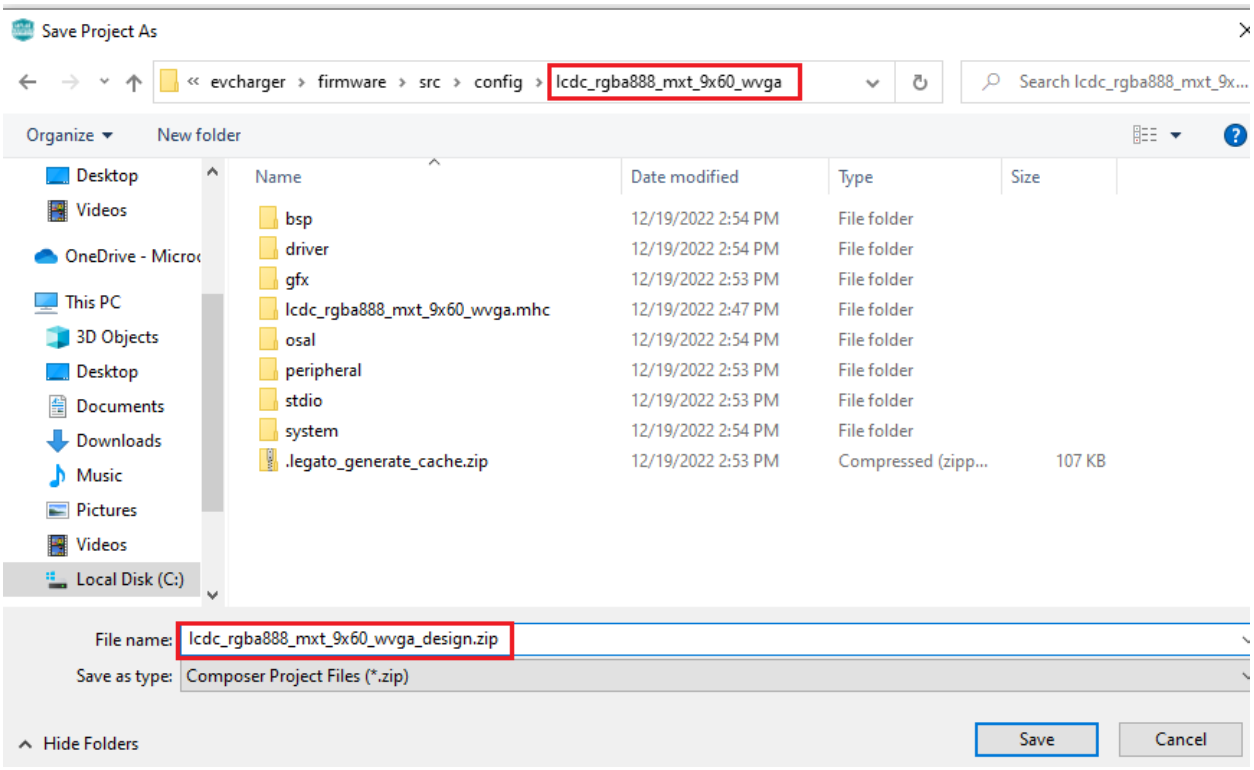



**To save and generate the design:**

ii. From the Composer main menu, click on File and Save:




iii. In the project src/config/<config_name>/ folder save the design project zip file by providing file name lcdc_rgba888_mxt_9x60_wvga_design:

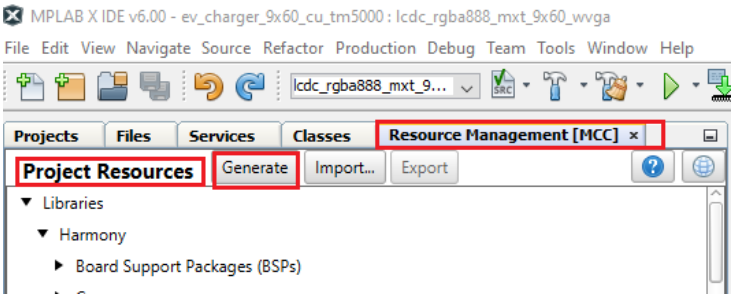



***Provide the file name as <config_name>_design. This ensures that the next time the MGC plugin is launched this design file is automatically launched. Otherwise, you will have to manually choose the design file to open.***

iv. Click on the Generate code icon  from MGC Suite.

***This completes our design for the first task.***

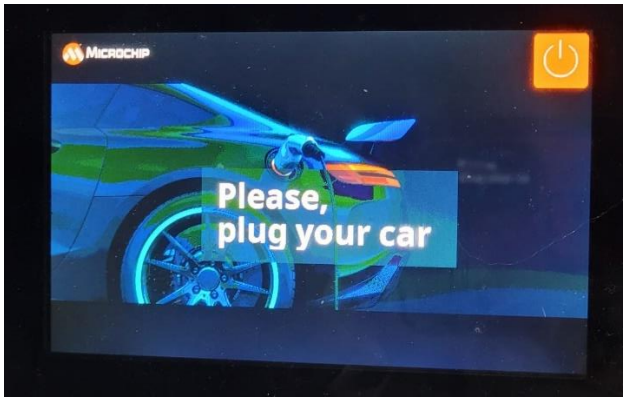
20. Going back to the MPLABX IDE, in the MCC Project Resources window, click “Generate”:




This updates the project with the graphics drivers for the design you just created.

21. Debug the project by clicking on the “Debug Project” icon  .

***You should see the following displayed on the LCD screen:***




***You now know how to create a graphics project with MCC & Composer on the MPLAB X IDE, build it and debug it on the MPU32 target.***

**Bonus Task**

Replace the square buttons:


button_off.png


button_on.png

With round buttons:


round_power_off.png


round_power_on.png

# Task 2: Managing Events & Screen Transition

## *Purpose:*

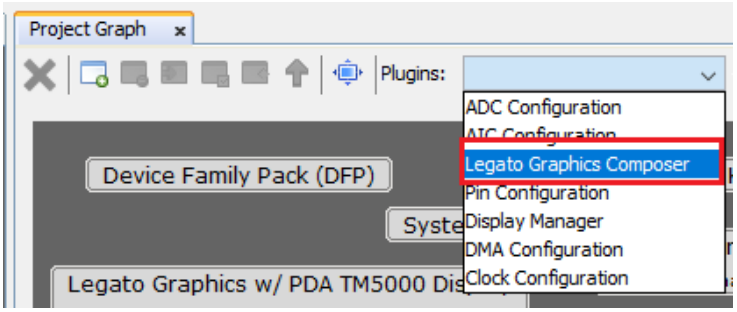
To learn how to manage events and screen transitions.

## *Overview:*

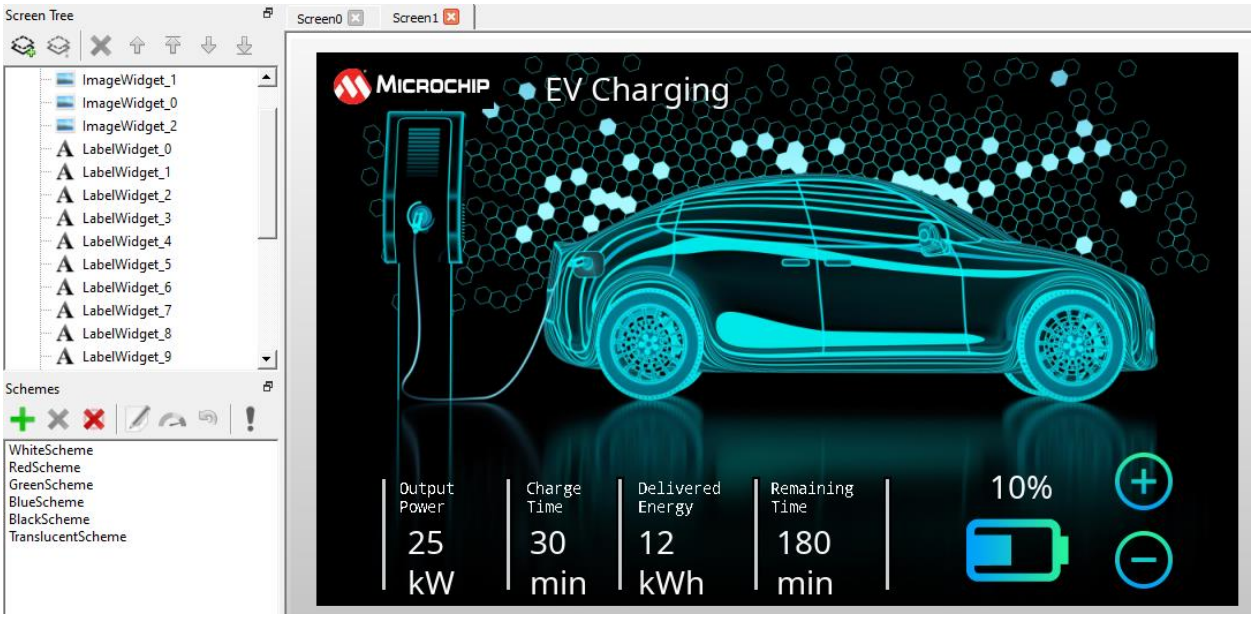
In this task we will discuss event management and screen transitions.

## *Procedure:*

1. In this task we provide the design file (.zip) for your convenience. Copy lcdc_rgba888_mxt_9x60_wvga_design.zip available to you in resources/task2 folder to your project source folder - <path>\evcharger\firmware\src\config\lcdc_rgba888_mxt_9x60_wvga (Replace the existing design file generated at the end of Task 1, you may backup this .zip file if you choose to do so). Launch the Microchip Graphics Composer from the plugins dropdown menu:

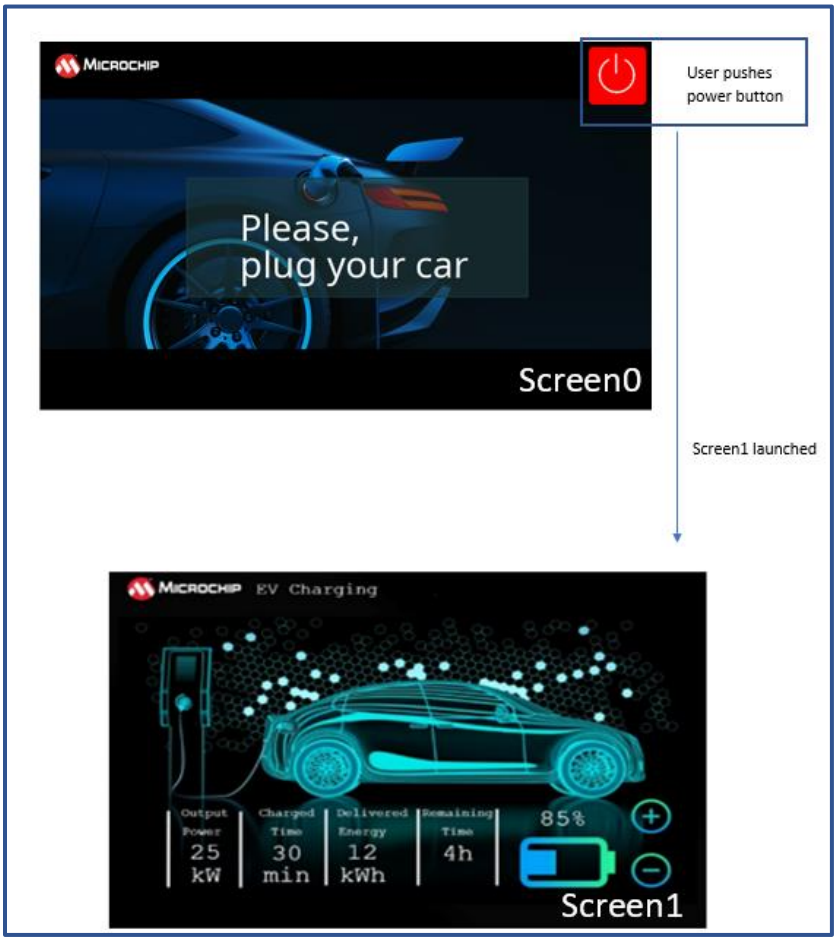



The MGC Suite is automatically launched with the design completed for this task.



***For a detailed description of the procedure to be followed to create the design yourself, please refer to Appendix 1.***

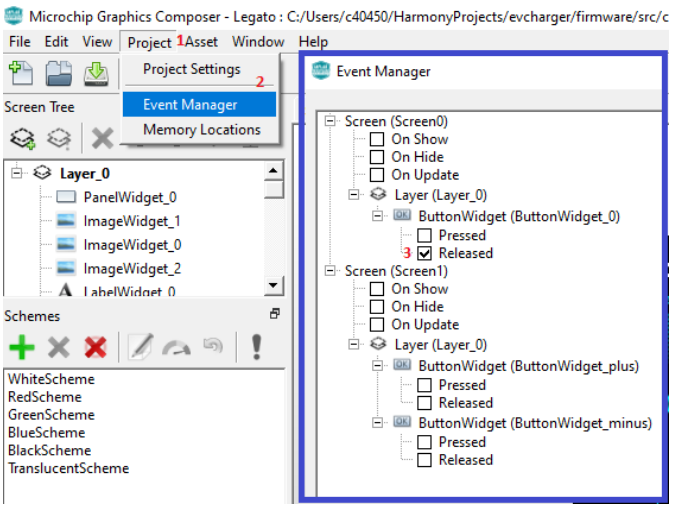
We need to display Screen1 when user presses the power button on Screen0:



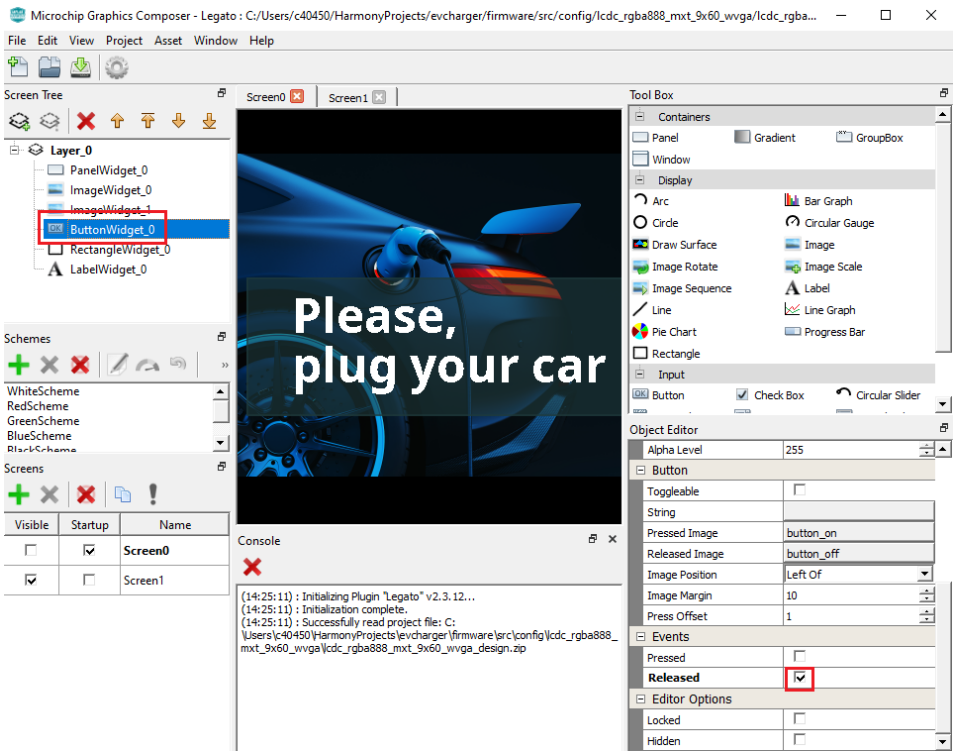

To do this we have to implement event handling and screen transition. Let us learn how to do implement a button event handler.

Button Event Callback

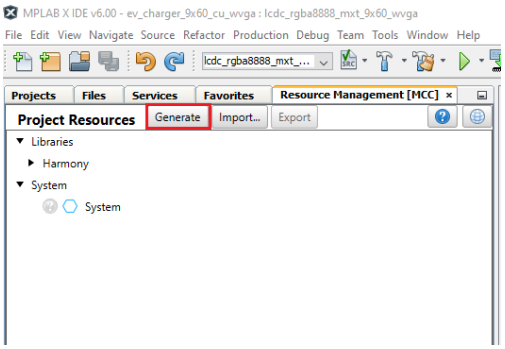
1. From the Composer main menu, select Project -> Event Manager and select the “Released” Checkbox for ButtonWidget.




***You can also select the button widget in the layer tree and from the object editor, select "Released" for Events:***




2. With Legato Composer open, go back to MCC tool on MPLABX IDE on "Generate" button on "Project Resources" pane.




***If you wish to close the Legato Composer, please remember to generate your design file and save your design by going to File->Save from the main menu.***

This adds the code as shown below in the following files in your project (path: evcharger\firmware\src\config\lcdc_rgba888_mxt_9x60_wvga\gfx\legato\generated\screen):

i. In le_gen_screen_Screen0.h:

```
// event handlers
// !!THESE MUST BE IMPLEMENTED IN THE APPLICATION CODE!!
void event_Screen0_ButtonWidget_0_OnReleased
(leButtonWidget* btn);
```

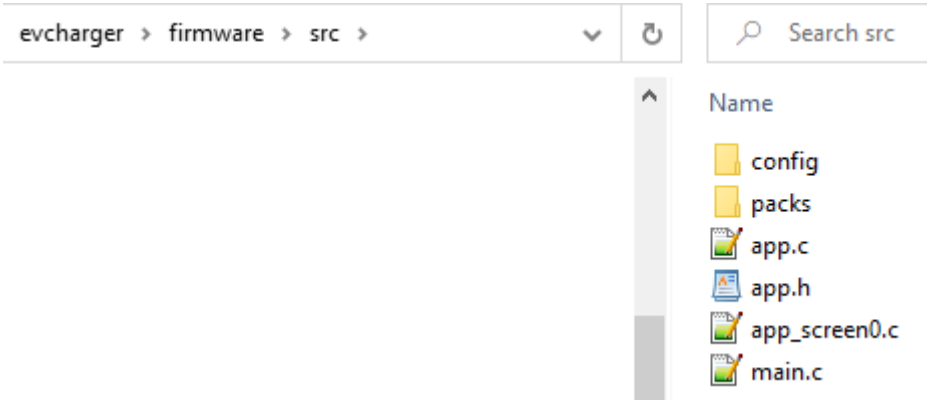
ii. In le_gen_screen_Screen0.c:

```
Screen0_ButtonWidget_0->fn-
>setReleasedEventCallback(Screen0_ButtonWidget_0,
event_Screen0_ButtonWidget_0_OnReleased);
```

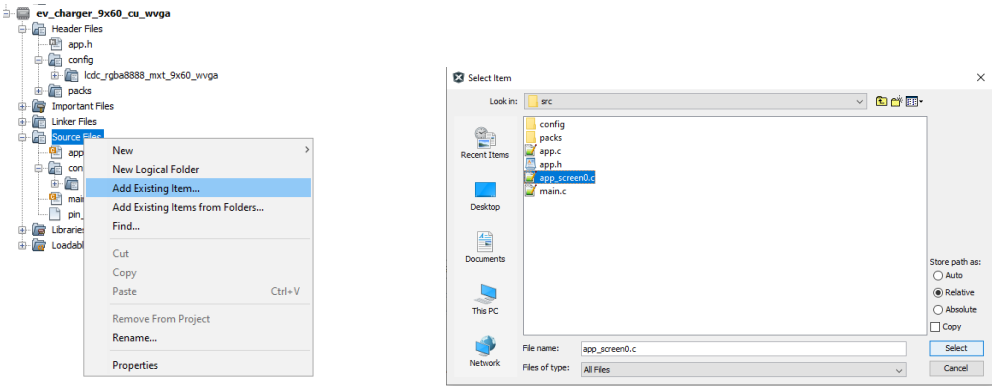
We need to implement the “event_Screen0_ButtonWidget_0_OnReleased (leButtonWidget* btn)” function.

It is recommended to create separate application file to handle each screen, especially for event handling and timers to avoid accessing widgets that are not on that screen and to keep your code organized.

3. In “<path>\evcharger\firmware\src” folder create “app_screen0.c”.



Right click on “Source Files” and select “Add Existing Item…” and choose app_screen0.c.

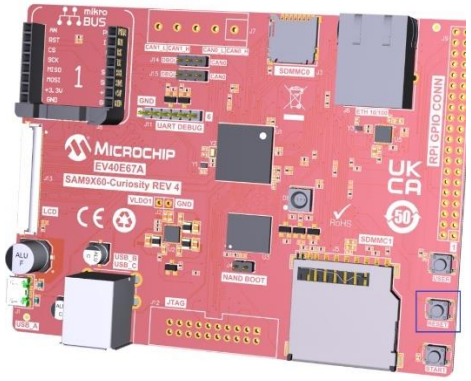



In app_screen0.c, we will implement the event_Screen0_ButtonWidget1_OnReleased() function. Add the following code to app_Screen0.c:

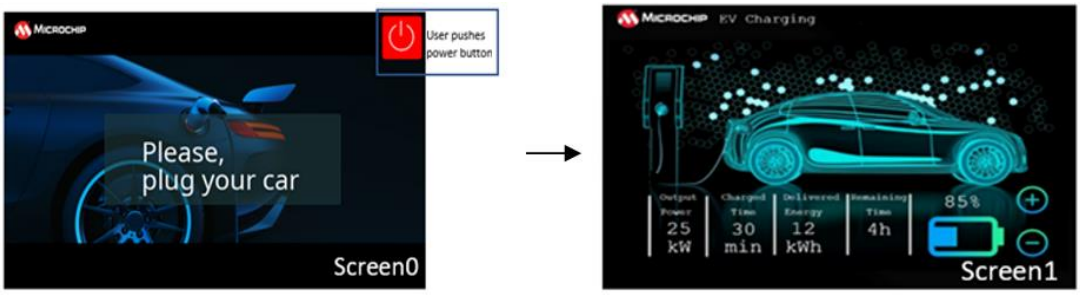
```
#include "definitions.h"

/* Event handler for ButtonWidget1: Launch Screen1 */
void event_Screen0_ButtonWidget_0_OnReleased(leButtonWidget* btn)
{
    legato_showScreen(screenID_Screen1);
}
```

The `legato_showScreen(screenID_Screen1)` function displays Screen1 defined by screenID_Screen1 in evcharger\firmware\src\config\lcdc_rgba888_mxt_9x60_wvga\gfx\legato\generated\le_gen_init.h.

4. Now press the “Reset” button on the SAM9X60 Curiosity board (highlighted in the picture blow) and click the “Debug Main Project” icon:  .




***You should see Screen0 displayed on the screen. On pressing the power button on the top right of the screen, you should see Screen1.***




***In this task we have seen how to handle screen transitions and Button event callback functions.***

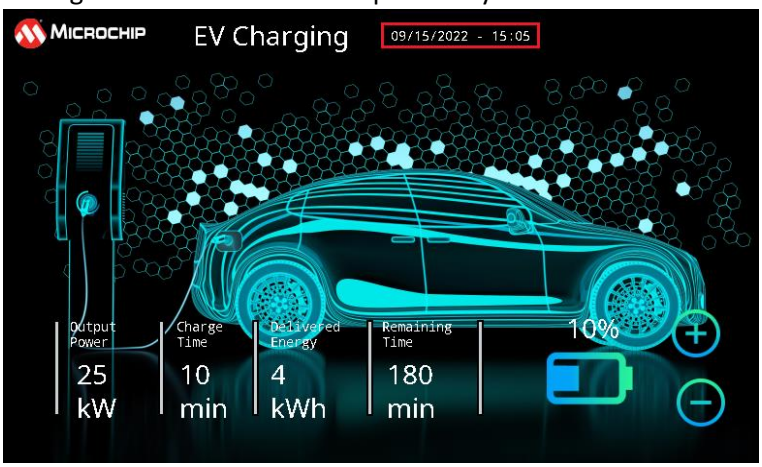
# Task 3: Dynamic String and RTC

## *Purpose:*

To learn how to dynamically manage and display strings using Timers.

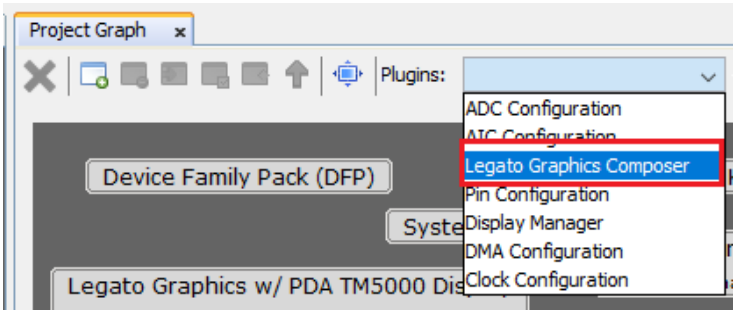
## *Overview:*

In this task we will dynamically change the contents of the Date/Time string using RTC as highlighted in the Figure below. The label is updated by the minute.




## *Procedure:*

1. The design file for this task is provided for you in the resources/task3 folder. Copy the lcdc_rgba888_mxt_9x60_wvga_design.zip file to your project source folder - <path>\evcharger\firmware\src\config\lcdc_rgba888_mxt_9x60_wvga (Replace the existing design file, you may backup the existing .zip file before replacing it if you choose to do so). Launch the Microchip Graphics Composer from the plugins dropdown menu:

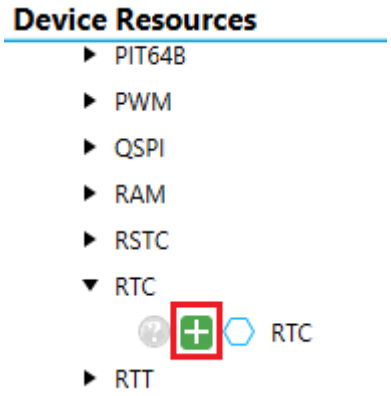



The MGC Suite is automatically launched with the design completed for this task.

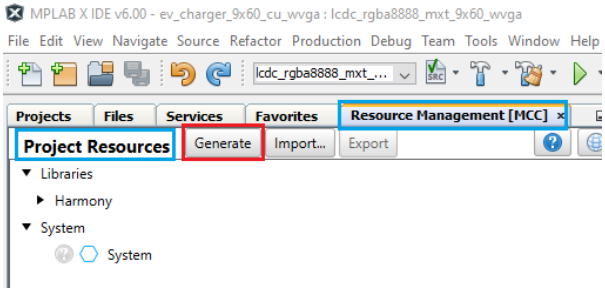
***For a detailed description of the procedure to be followed to create the design yourself, please refer to Appendix 2.***

Adding RTC support using MCC

2. From “Device Resources” pane, expand Libraries -> Harmony -> Peripherals -> RTC. Click on green button as shown below which adds RTC to the project graph:

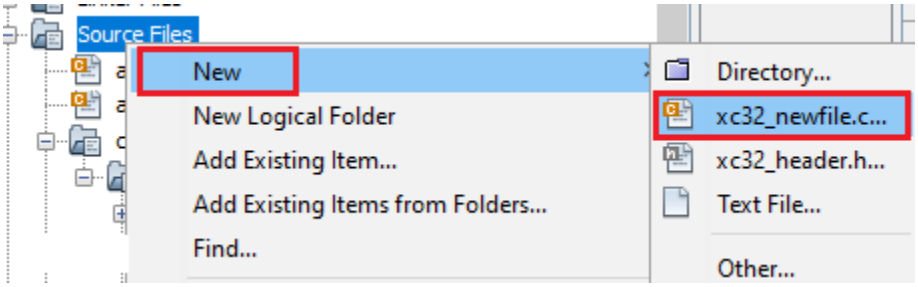



3. Then from MCC Resources Management pane, click “Generate” button.

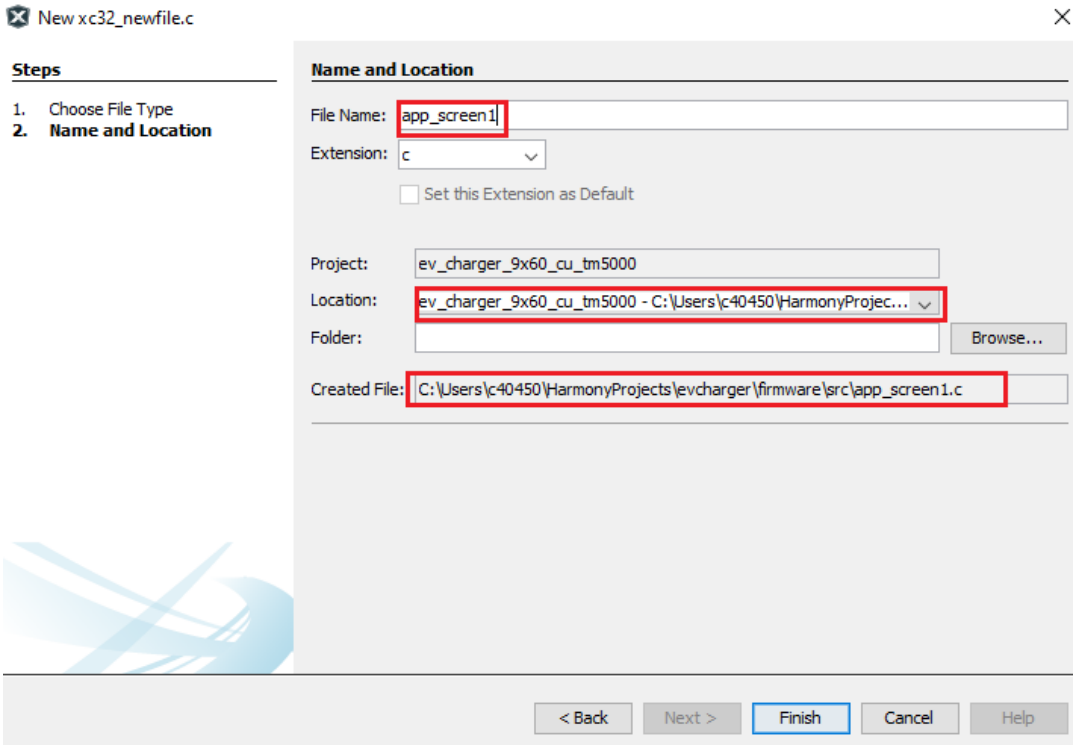



Writing Application Code

4. Right click on “Source Files” and click on New -> x32_newfile.c.




Provide file name as “app_screen1” as shown below:




5. Delete the populated code in app_screen1.c and include the relevant header files and required variables as shown below:

```
#include <stdio.h>
#include "gfx/legato/generated/le_gen_init.h"
#include <time.h>
#include "gfx/legato/generated/screen/le_gen_screen_Screen0.h"
#include "config/lcdc_rgba888_mxt_9x60_wvga/peripheral/rtc/plib_rtc.h"
#include "app.h"

#define MAX_TIME_STRING_LEN     18

// Structure to store RTC data
struct tm currentTime;
//variable to keep track of when to update time
int lastminute;
//Legato string object
leFixedString p_timestring;

//Legato Char buffer
static leChar p_legatoTimeBuff[MAX_TIME_STRING_LEN] = {0};

//C character buffer
static char p_timecharbuff[MAX_TIME_STRING_LEN];
```

6. Next in app_screen1.c, add a function to initialize these variables (**`init_RTC_Label_Screen1`**) and a function to be called periodically to update the date/time label (**`UpdateTime_Label`**):

```
void init_RTC_Label_Screen1(void)
{
    lastminute=0;
    leFixedString_Constructor(&p_timestring,  p_legatoTimeBuff, MAX_TIME_STRING_LEN
*2);
    p_timestring.fn->setFont(&p_timestring, (leFont*)& NotoMono14);

}


void UpdateTime_Label (void)
{
    RTC_TimeGet( &currentTime );
    if(lastminute != currentTime.tm_min)
    {
        lastminute = currentTime.tm_min;
        memset(p_timecharbuff,0,sizeof(p_timecharbuff));
        sprintf(p_timecharbuff,"%02d/%02d/%04d - %02d:%02d",1+currentTime.tm_mon,
currentTime.tm_mday,1900+currentTime.tm_year,
currentTime.tm_hour,currentTime.tm_min);
        p_timestring.fn->setFromCStr(&p_timestring, p_timecharbuff);

        Screen1_rtc_label->fn->setString(Screen1_rtc_label,
(leString*)&p_timestring);
    }
}
```

7. Declare the init_RTC_Label_Screen1 and UpdateTime_Label functions in app.h header file.
8. Now in the APP_Tasks() function, in APP_STATE_INIT case, call the `init_RTC_Label_Screen1` function:

```
switch ( appData.state )
    {
        /* Application's initial state. */
        case APP_STATE_INIT:
        {
            bool appInitialized = true;
            init_RTC_Label_Screen1();
            if (appInitialized)
            {

                appData.state = APP_STATE_SERVICE_TASKS;
            }
```

9. In APP_Tasks() function in app.c, add the code to update the time string if application is in Screen1 context in APP_STATE_SERVICE_TASKS case:

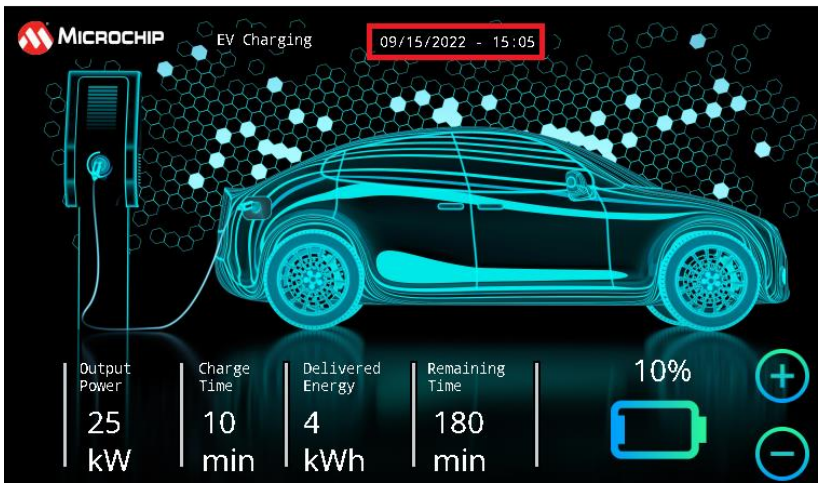
```
       case APP_STATE_SERVICE_TASKS:
               {
                   if(legato_getCurrentScreen()==screenID_Screen1)
                       UpdateTime_Label();

                   break;
        }
```

10. In app.c include the following header file:

```
#include "definitions.h"
```

11. Now press the “Reset” button on the SAM9X60 Curiosity board and click the “Debug Main Project” icon:  .

***You should see the date and time displayed on the LCD screen and the time is updated by the minute.***




***You now know how to display dynamic strings.***

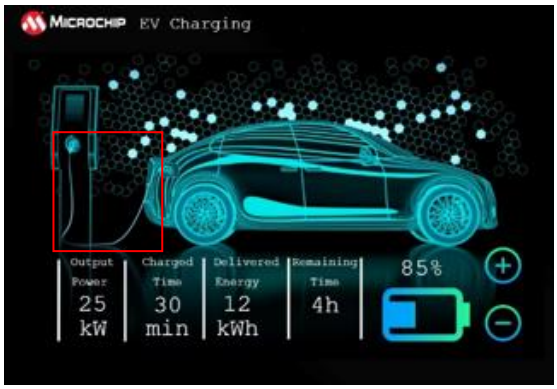
# Task 4: Sprite Animation using Canvas

## *Purpose:*

To learn how to use Graphics Canvas.

## *Overview:*

In this task we will learn how to create sprite effect animations using Canvas. In Screen2 we will add sprite effect animation to the charge cable in the background image:

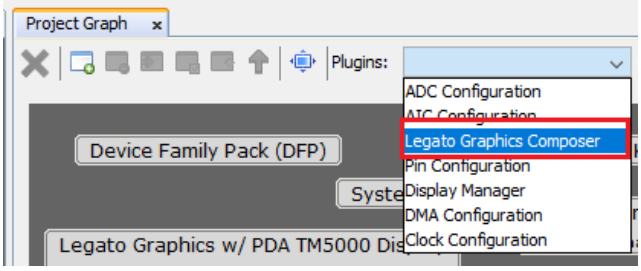



We will also discuss the importance of using the different Layers of the LCD Controller in our design to improve performance.

## *Procedure:*

<u>Changes in MGC</u>

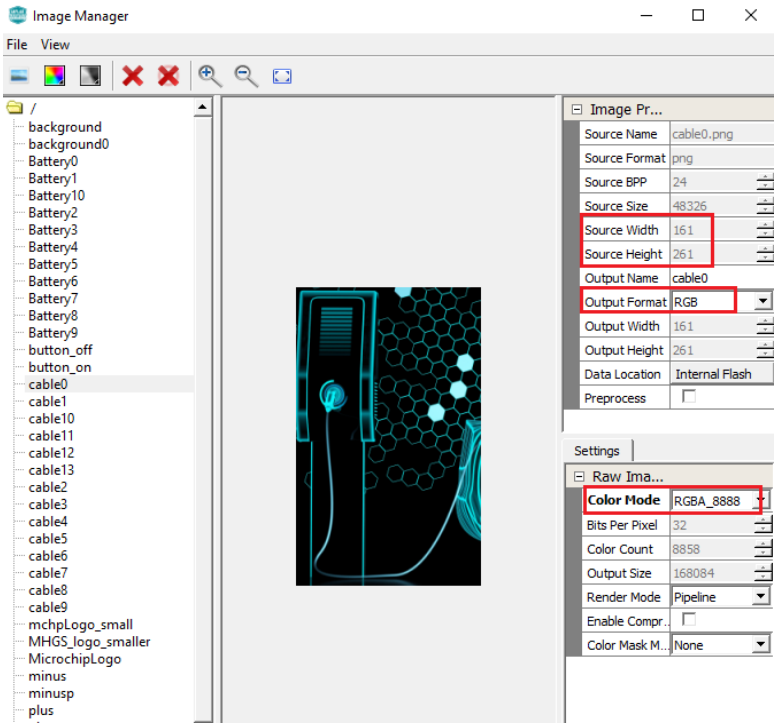
1. The design file for this task is provided for you in the resources/task4 folder. Copy the lcdc_rgba888_mxt_9x60_wvga_design.zip file to your project source folder - <path>\evcharger\firmware\src\config\lcdc_rgba888_mxt_9x60_wvga (Replace the existing design file, you may backup the existing .zip file before replacing it if you choose to do so). Launch the Microchip Graphics Composer from the plugins dropdown menu:



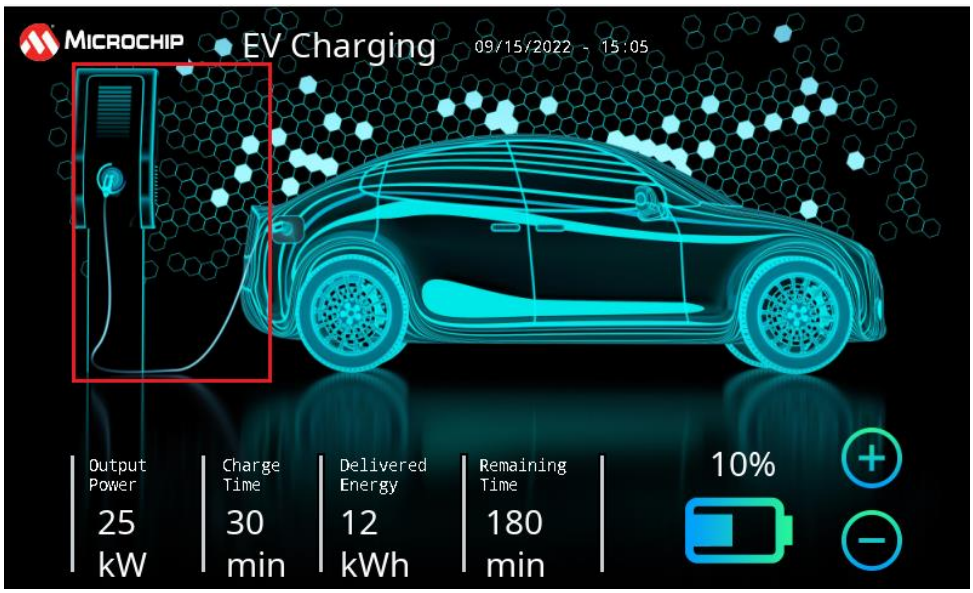

The MGC Suite is automatically launched with the design completed for this task.

***Launch "Image Manager" to check that 14 background images are added to the project (cable0 – cable13). Their Output format is set to RGB and Color Mode to "RGBA_8888". The "Enable Compression" is not set.***

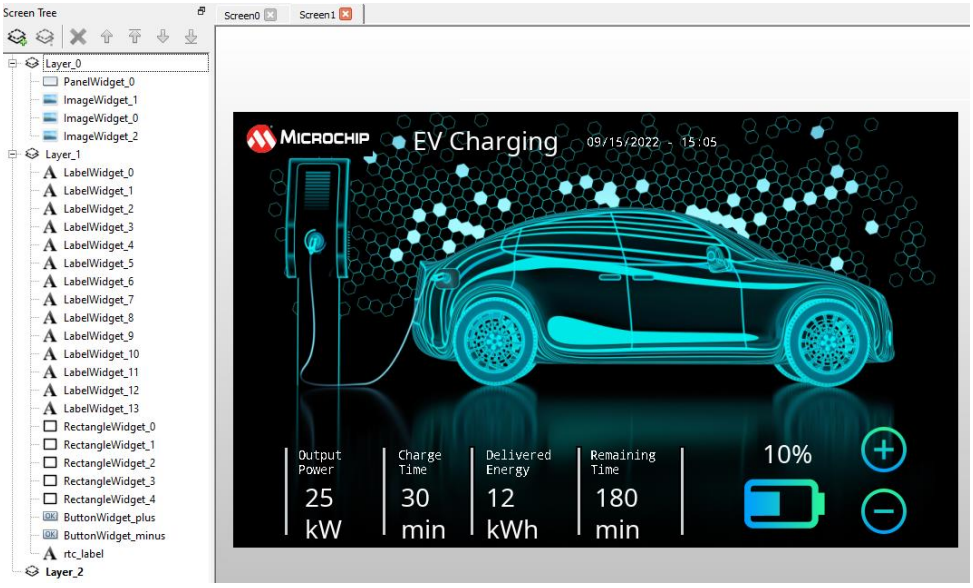
***Make sure you DO NOT check the "Enable Compression" checkbox. Canvas requires that images used be uncompressed raw images.***



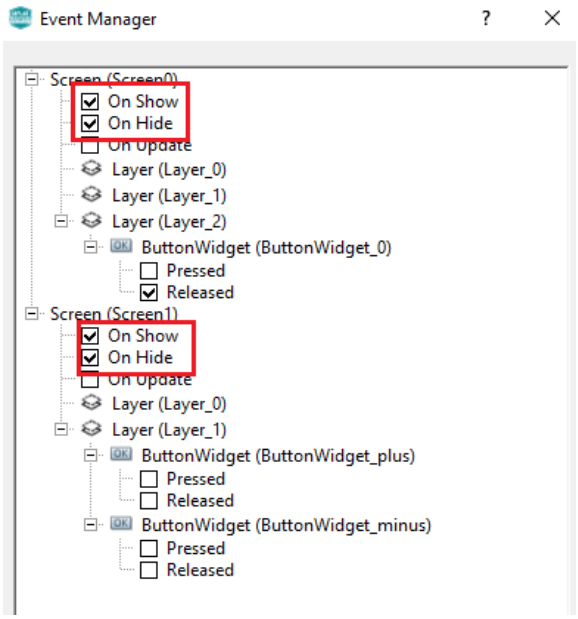

***Please take a moment to look at these images. They will be superimposed on the background image at the location 58x48 and the size of these images is 161x261 (pixels):***




2. Also observe that the static assets are arranged in lower layer (layer 0) and assets that change more rapidly are placed in higher layer (layer 1). We have seen (in the lecture section) distributing assets this way between layers makes rendering more efficient and faster. We will user layer 2 for sprite animation of the cable.




3. From the Event Manager click on the checkbox for Screen0 and Screen1 “On Show” and “On Hide” events:

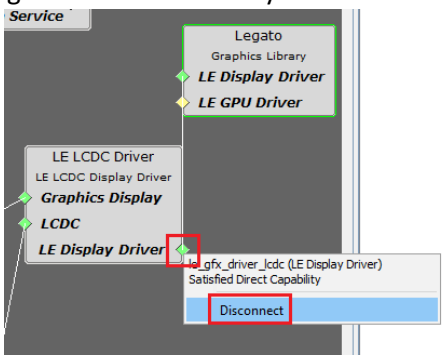



The Screen1 OnShow event will be used to start the TC0 timer which will increment a tick count. Each tick count increment cycles through the sprite sheet of images. The Screen1 OnHide event will be used to stop the TC0 timer.
Screen0 OnShow is used to display the Screen0 assets using Canvas.

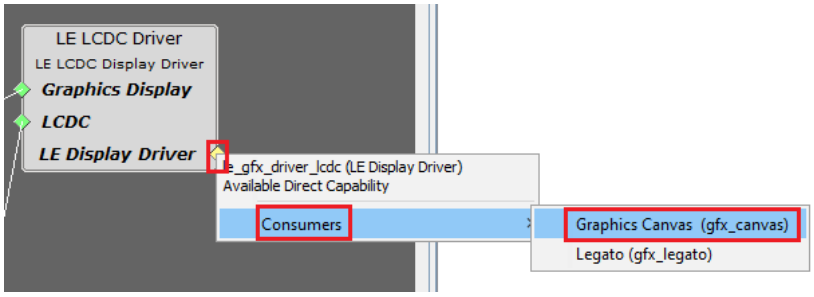
Adding Canvas Support in MCC

Let us create the sprite animation effect using Graphics Canvas which is a feature supported by the LCDC driver. To add this support, do the following:

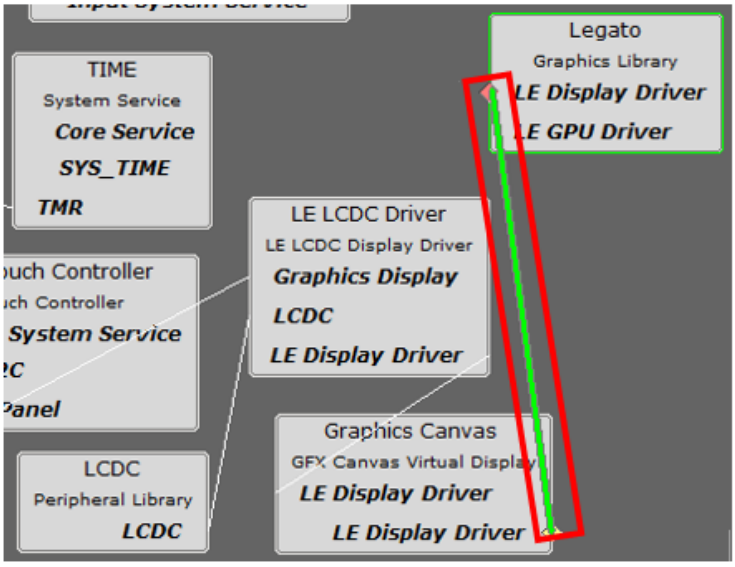
4. Right click on the green diamond on the “LE LCDC Driver” component and click on “Disconnect”. The green diamond turns yellow.



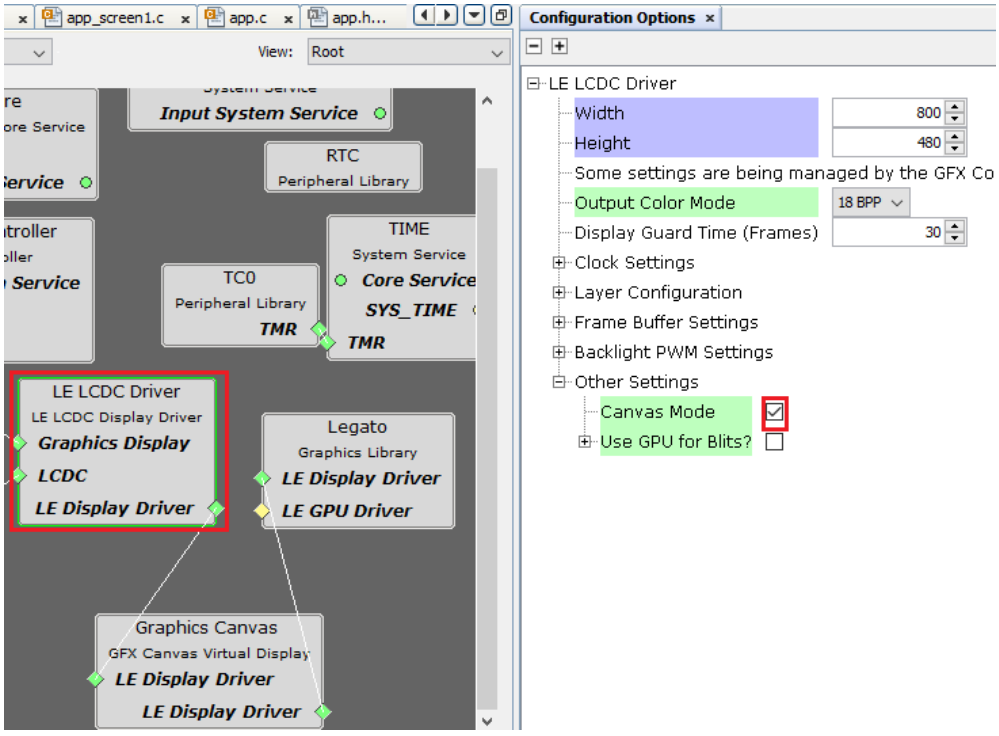

Now right-click on the yellow diamond on the “LE LCDC Driver” component and for Consumers, click on “Graphics Canvas”.



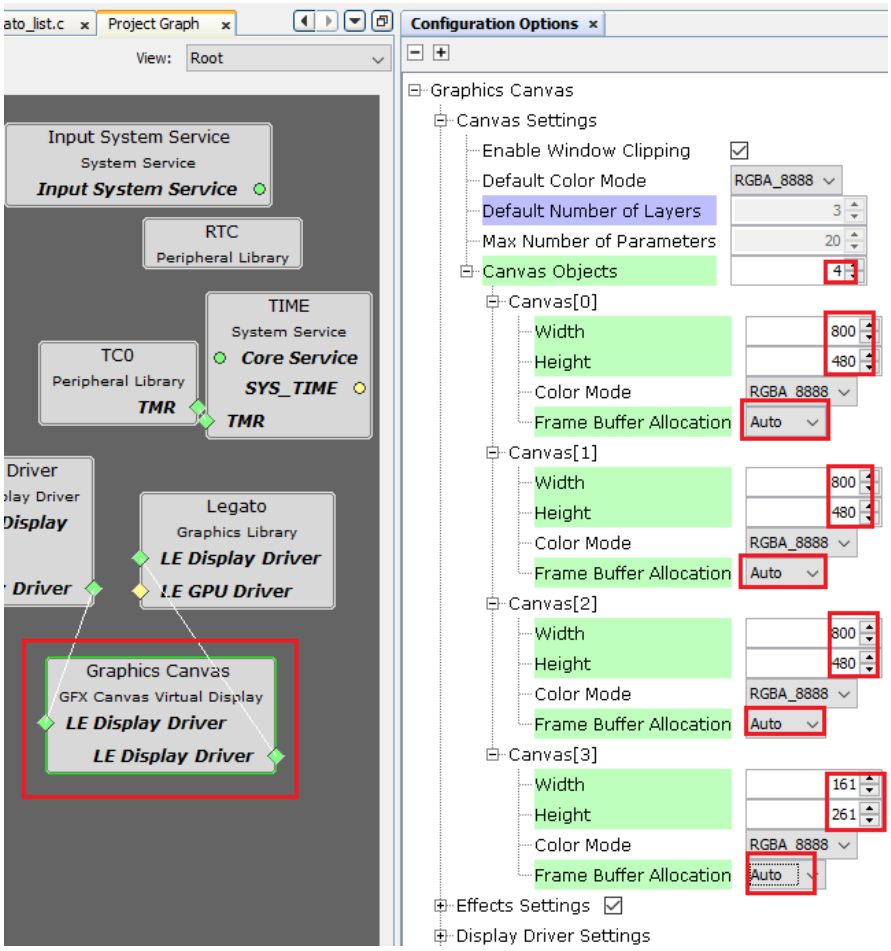

Connect “LE Display Driver” on the “Graphics Canvas” component and “Legato” component.




5. Click on “LE LCDC Driver” component and from the Configuration window, select “Canvas Mode” checkbox from “Other Settings”.

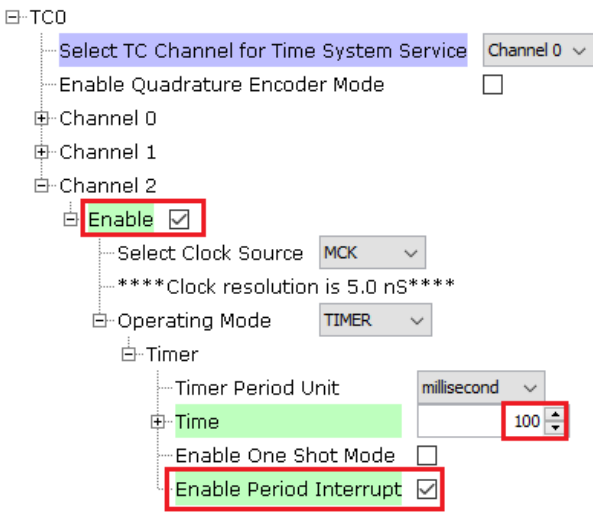


6. Click on “Graphics Canvas” component and from the configuration window set the options as shown below:



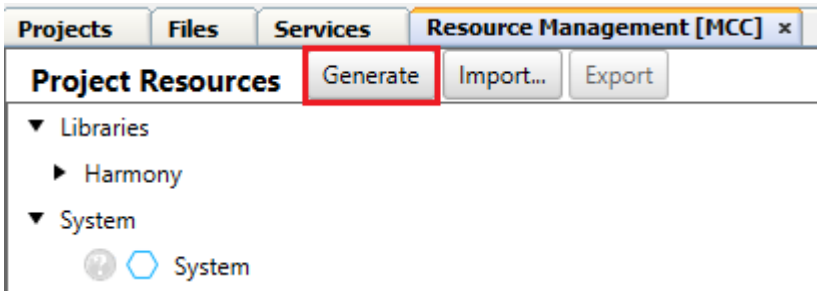

***Canvas0, Canvas1 and Canvas2 buffers are for the 3 LCD layers (Layer0, Layer1 and Layer2). Canvas3 buffer is to animate the charge cable with dimension 161x261 at pixel location 58x48 (as explained before).***

7. Click on TC0 click on TC0 Channel 2, select “Enable Period Interrupt” checkbox and Set Time for 100 ms:




This timer will be used to cycle through the sprite images using Canvas.

8. Regenerate code using MCC tool:




Application Code development

9. For your convenience, we have provided the application code required for this task. Replace the app.c, app.h, app_screen0.c and app_screen1.c in the MCC project with the ones provided in the resources/task4 folder.
   To review the code that has been added please refer to Appendix 3.
10. Now press the "Reset" button on the SAM9X60 Curiosity board and click the "Debug Main Project" icon:  .

***You should see the green charge cable sprite animation displayed on Screen1. You should also see the dynamic string updated by the minute.***

***You now know how to use Graphics Canvas for animation.***

# Task 5: Animation using Image Sequence Widget

## *Purpose:*

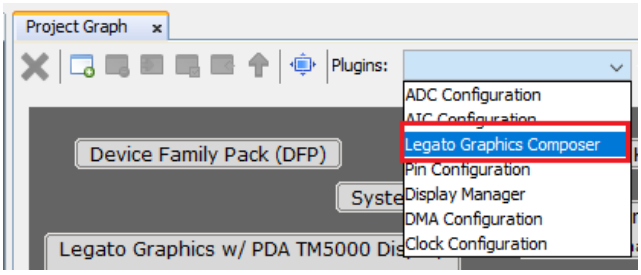
To learn how to animate images using Image Sequence Widget.

## *Overview:*

In this task we will learn how to create sprite animation effect using Image Sequence Widget. When the user presses the ⊕ and ⊖ buttons, the battery level image will change accordingly. Also, all the labels highlighted in the image below will change.


## *Procedure:*

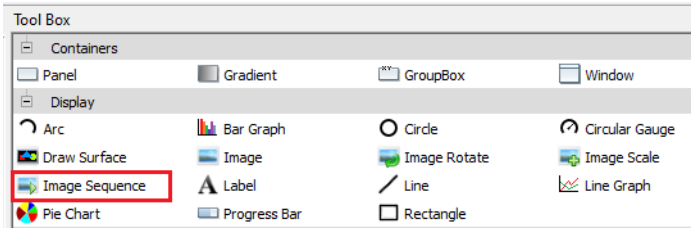
1. The design file for this task is provided for you in the resources/task5 folder. Copy the lcdc_rgba888_mxt_9x60_wvga_design.zip file to your project source folder - <path>\evcharger\firmware\src\config\lcdc_rgba888_mxt_9x60_wvga (Replace the existing design file, you may backup the existing .zip file before replacing it if you choose to do so). Launch the Microchip Graphics Composer from the plugins dropdown menu:

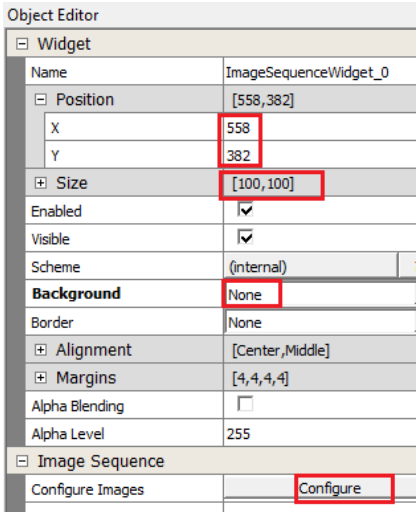


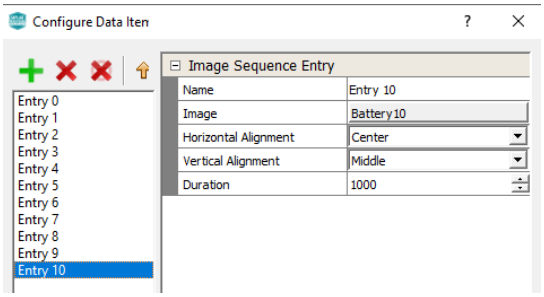

The MGC Suite is automatically launched with the design completed for this task.

***For a detailed description of the procedure to be followed to create the design yourself, please refer to Appendix 4.***

***You can see that ImageWidget2 (battery image) is removed from the design and is replaced with ImageSequence widget which is configured to cycle through 11 battery images.***

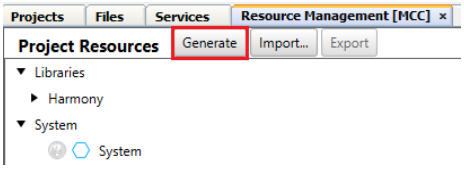



***The meter labels (shown below) are also changed when the ⊕ and ⊖ buttons are pressed with the help of application code provided to you.***


2. Regenerate code using MCC tool:




<u>Application Code development</u>

3. For your convenience, we have provided the application code required for this task. Replace the app_screen1.c in the MCC project with the one provided in the resources/task5 folder.

***To review the code that has been added please refer to [Appendix 5](#).***

4. Now rebuild and debug.

***You should see label widgets and battery image change with the ⊕ and ⊖ button press.***

***You now know how to use Image Sequence Widget.***

# Task 6: Animation using widget API

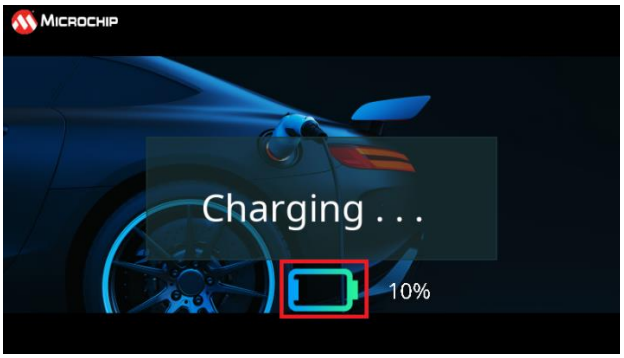
## *Purpose:*

To learn how to animate using Widget API - setWidth.

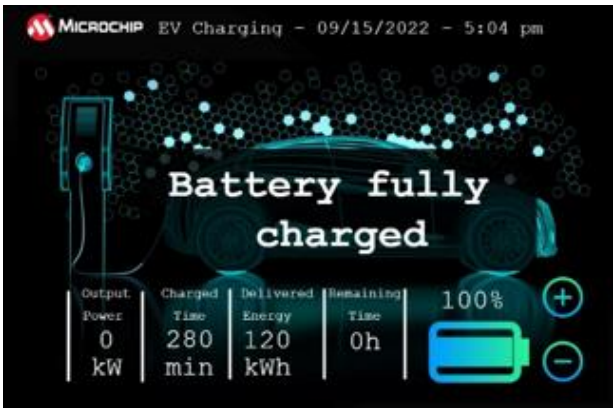
## *Overview:*

So far, we have learnt about animation using Canvas and Image sequence widget. In this task we will learn yet another simple yet effective way to animate using the widget API. We will learn how to animate the battery level using a panel widget using the setWidth API.

From Screen1_OnShow event, we will start a 10 second timer and if no buttons have been pressed in this duration Screen0 is displayed with the battery level animation (Charging Screen):

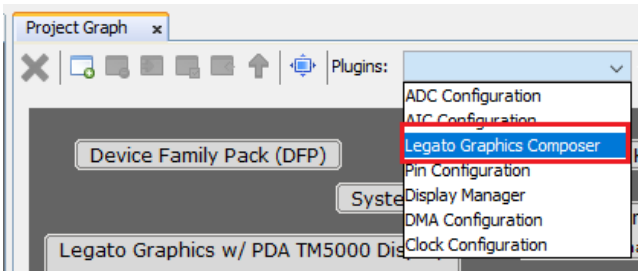



After 10 s, the following screen is displayed (Charged Screen):




## *Procedure:*

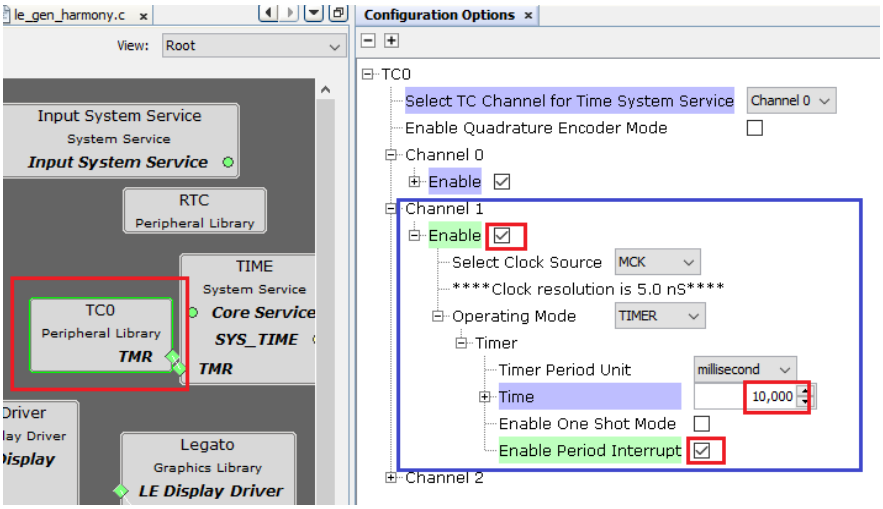
1. The design file for this task is provided for you in the resources/task6 folder. Copy the lcdc_rgba888_mxt_9x60_wvga_design.zip file to your project source folder - <path>\evcharger\firmware\src\config\lcdc_rgba888_mxt_9x60_wvga (Replace the existing design file, you may backup the existing .zip file before replacing it if you choose to do so). Launch the Microchip Graphics Composer from the plugins dropdown menu:




The MGC Suite is automatically launched with the design completed for this task.

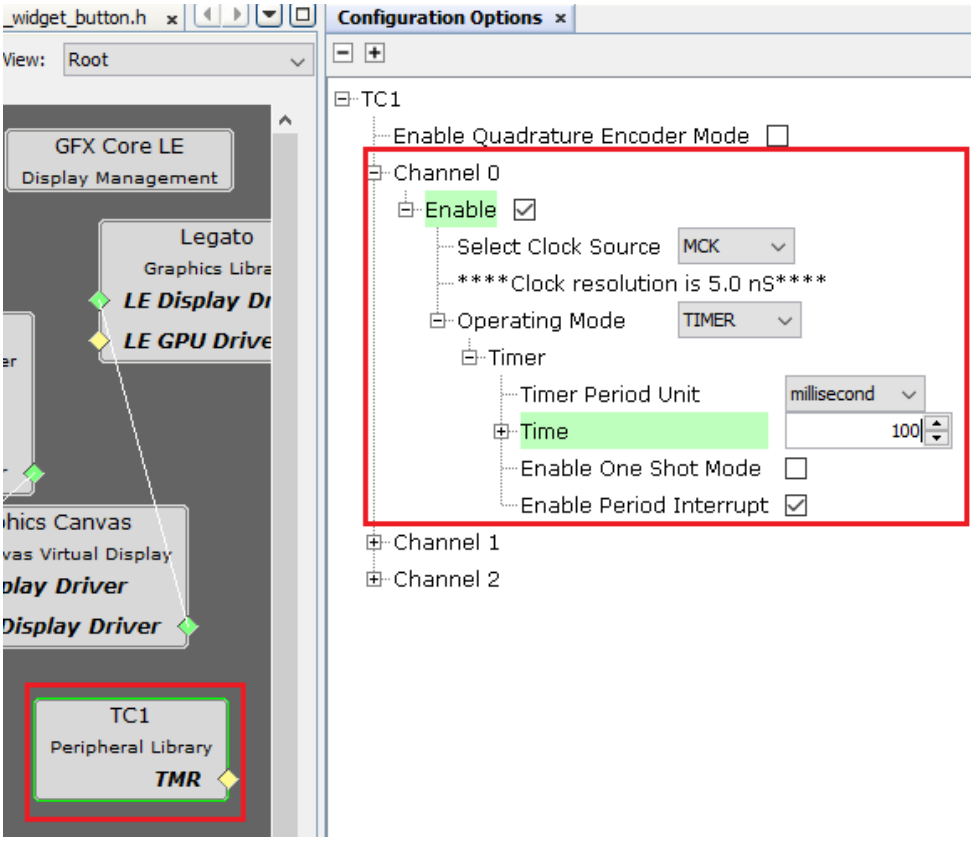
***For a detailed description of the procedure to be followed to create the design yourself, please refer to Appendix 6.***

2. From the project graph, choose TC0 and from the configuration window, set the following options:




***TC0_Channel 1 is used to keep track of inactivity in Screen1 so that we can show the charging screen (Screen0).***

3. We need another time (TC1) to keep track of the battery animation in Screen0 “Charging Screen”.
   From the MCC “Device Resources” Panel, expand Harmony -> Peripherals-> TC -> TC1. Click on the icon to add TC1 support to the project graph. Select TC1 component and sets its options using the Configuration window as shown below:




4. Regenerate code using MCC tool.

Adding Application Code

5. For your convenience, we have provided the application code required for this task. Replace the app.c, app.h, app_screen0.c, app_screen1.c in the MCC project with the one provided in the resources/task6 folder.

***To review the code that has been added please refer to Appendix 7.***

6. In app_screen0.c, there is a function that updates the performs battery animation on Screen0 when the application is in DEMO_SCREEN1 for SCREEN_STATES (called by APP_Tasks):

```
void Update_Screen2(void)
{
    if (battery_tick != prev_batt_tick)
    {
        prev_batt_tick = battery_tick;

        if(i<=(percent_battery_charge/10)){
            /*Screen0_PanelWidget_battery->fn-
```

```
>setWidth(Screen0_PanelWidget_battery, (64*0.1*++i));*/
            gfxcShowCanvas(LAYER_2_CANVAS_ID);
            gfxcCanvasUpdate(LAYER_2_CANVAS_ID);
        }else{
            i=0;
        }
    }
    if(sec_cntr >= 10){
        TC1_CH0_TimerStop();
        legato_showScreen(screenID_Screen1);
    }
}
```

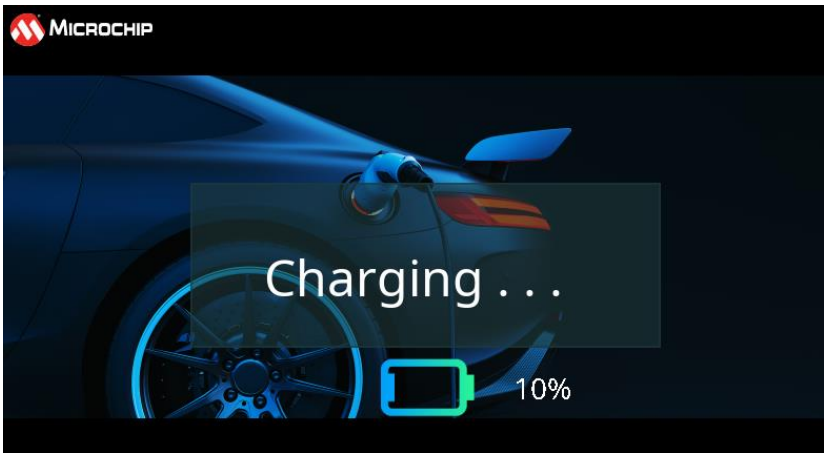
Remove comments from the following line of code:

`Screen0_PanelWidget_battery->fn->setWidth(Screen0_PanelWidget_battery, (64*0.1*++i));`

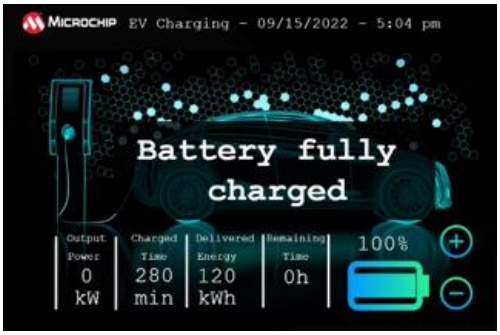
The panel widget (PanelWidget_battery) width is changed using the `setWidth` API periodically to cause animation effect.

7. Now rebuild and debug.

***From Screen1 if the ⊕ and ⊖ buttons are not pressed for 10s, you should see Screen1 display the following with the battery level animation using the setWidth API of the panel widget and the label displaying the chosen battery charge percentage:***




***Once 10s have passed Screen1 is displayed back:***




***You now know how to use panel widget API to create animation effects.***

# Task 7: Running The Final Application

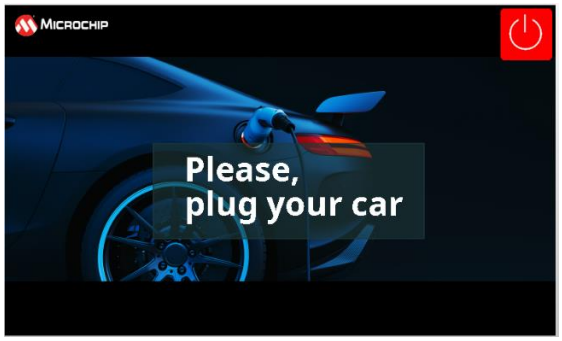
## *Purpose:*

To run the final EV Charger demonstration application on the SAM9X60 Curiosity board.

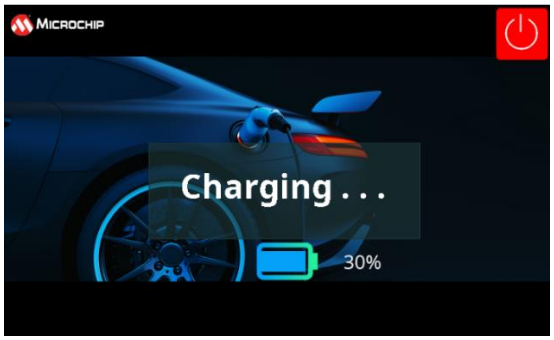
## *Overview:*

So far, we have learnt different concepts in Harmony Graphics such as Layers, Canvas, Schemes, Widgets, Dynamic Strings and Event Handling. However, the application we have developed so far needs some minor modifications to meet the specifications of the final application. Also, we see that screen transition is quite slow. In this task we will:

- Design 3 layers
    - Layer 1 will contain all assets for Scene 0 and Scene 2:

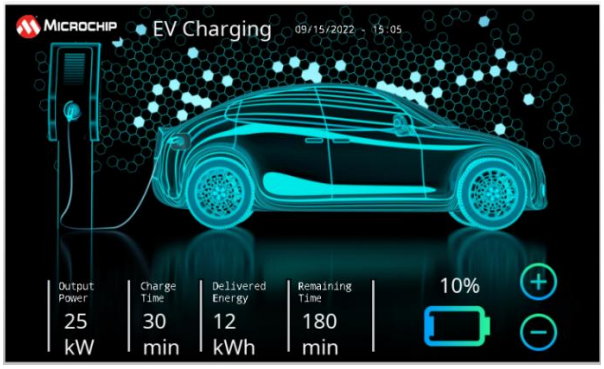



Scene 0




Scene 2

    - Layer 2 will contain all assets for Scene 1:



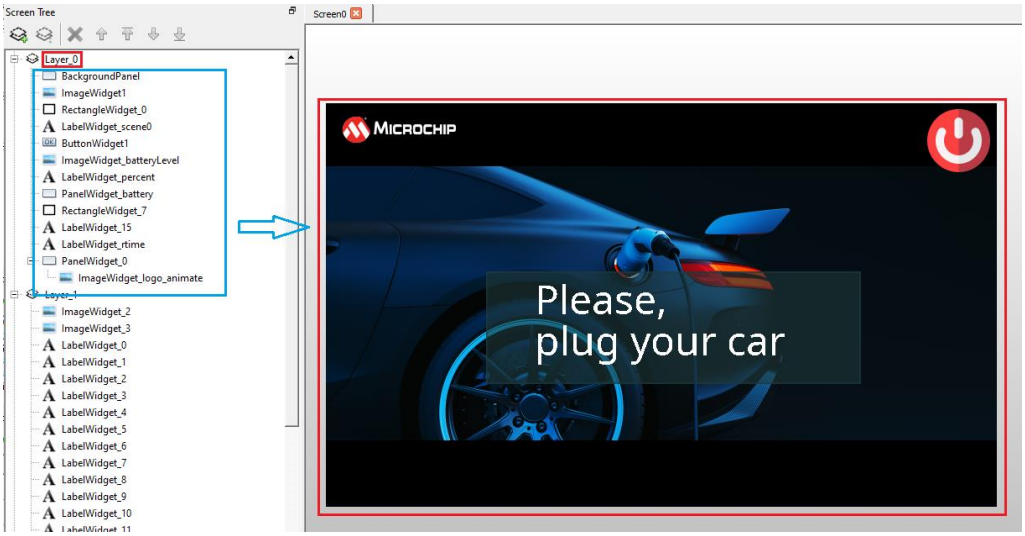

Scene 1

    - Layer 3 will be used for cable animation
- We will define 3 Canvas objects for the 3 layers and swap the Canvas using gfxcShowCanvas/ gfxcHideCanvas and gfxcCanvasUpdate functions to transitions between the scenes instead of doing an actual screen transition
- 10 s inactivity in Scene 1 causes a transition to Scene 2 (Canvas swap)
- We will use a button (user push button on the SAM9X60 Curiosity board) press event to transition from scene 2 to scene 1

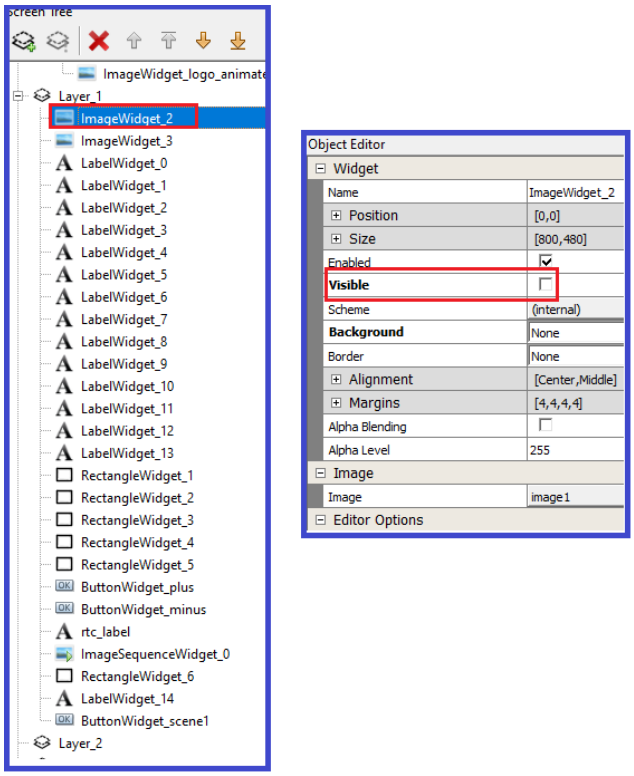
- In Scene 1, if the battery charge reaches 100% then “Battery Fully Charged Message” is displayed, and Scene 2 is never displayed
- We will add button widget to scene 1 to transition between scene 0 and scene 1
- We will rename the application files to app_scene0, app_scene1 and app_Scene2 to distinguish that we are not performing an actual screen transition which will yield in faster performance

## ***Procedure:***

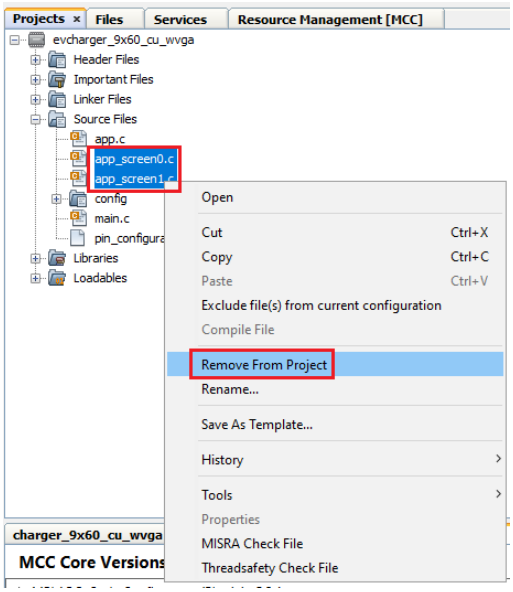
1. The design file for this task is provided for you in the resources/task7 folder. Copy the lcdc_rgba888_mxt_9x60_wvga_design.zip file to your project source folder - <path>\evcharger\firmware\src\config\lcdc_rgba888_mxt_9x60_wvga (Replace the existing design file, you may backup the existing .zip file before replacing it if you choose to do so). Launch the Microchip Graphics Composer from the plugins dropdown menu. The MGC Suite is automatically launched with the design completed for this task.
   - You can see that Layer_0 has all the assets to design Scene 0:



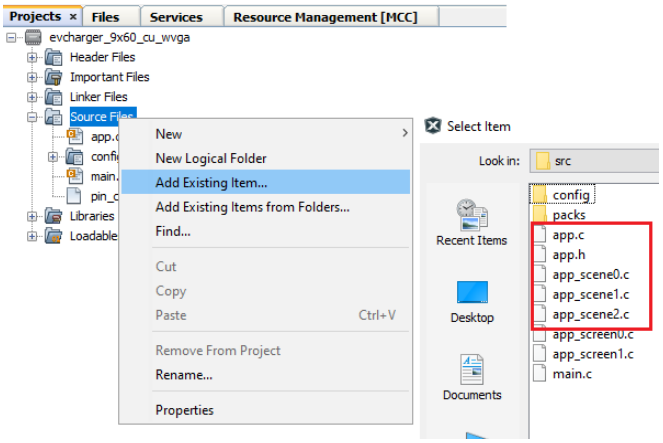

   - You can see that Layer_1 has all the assets to design Scene 1 in the same Screen (Screen 0) and it is set to invisible:



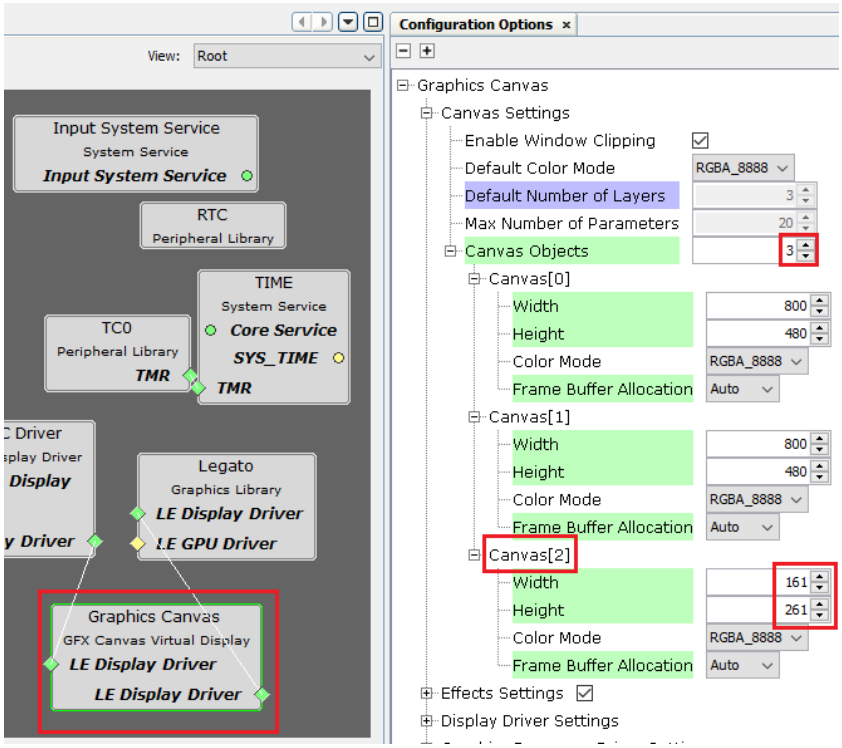

2. We have provided the application code required for this task. Copy the app.c, app.h, app_scene0.c, app_scene1.c, and app_scene2.c from the resources/task7 folder to your current project in the evcharger\firmware\src folder.
3. Right click on app_screen0.c and app_screen1.c in the project explorer and select “Remove From Project”.




4. Right click on the “Source Files” from the project explorer and select "Add Existing Item…”. Choose app.c, app.h, app_scene0.c, app_scene1.c, and app_scene2.c.




5. In the MCC Project Graph, choose “Graphics Canvas” component and change Canvas Objects to 3 and set Canvas[2] size to – 161x261 as shown below:




6. Regenerate code using MCC tool.
7. Rebuild and debug to see the desired behavior of the final EV Charger application.

***You have designed and developed an EV Charger demonstration application using Harmony Graphics.***

# Conclusion

In this training we have developed a simple EV Charger demonstration application. In doing so we have learned the following topics:

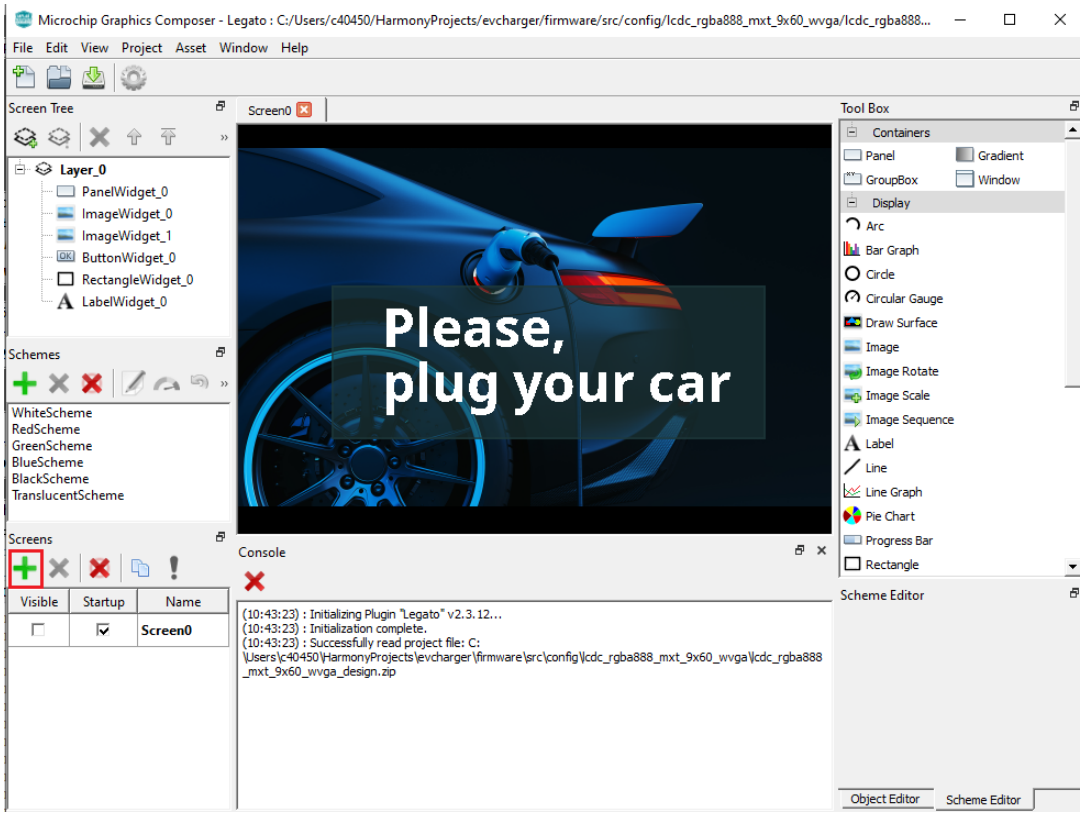
1. Adding Images, fonts, strings using Asset Manager
2. Handling simple widgets - Buttons, labels, images, image sequence widget
3. Modifying the look and feel of the widget using schemes
4. Multiple layers in graphics design and its importance
5. Button events and screen transitions, screen (OnShow/OnHide) events
6. Timers & RTC
7. Dynamic Strings
8. Graphics Canvas and sprite animation

We also discussed some pro-tips to make the application perform better using Canvas.

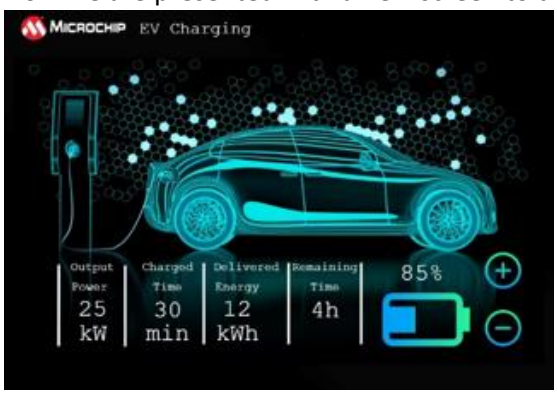
# Appendix 1 – Task 2

Adding New Screen

1. In legato composer, from the “Screens Manager” click on the + sign to add new screen:



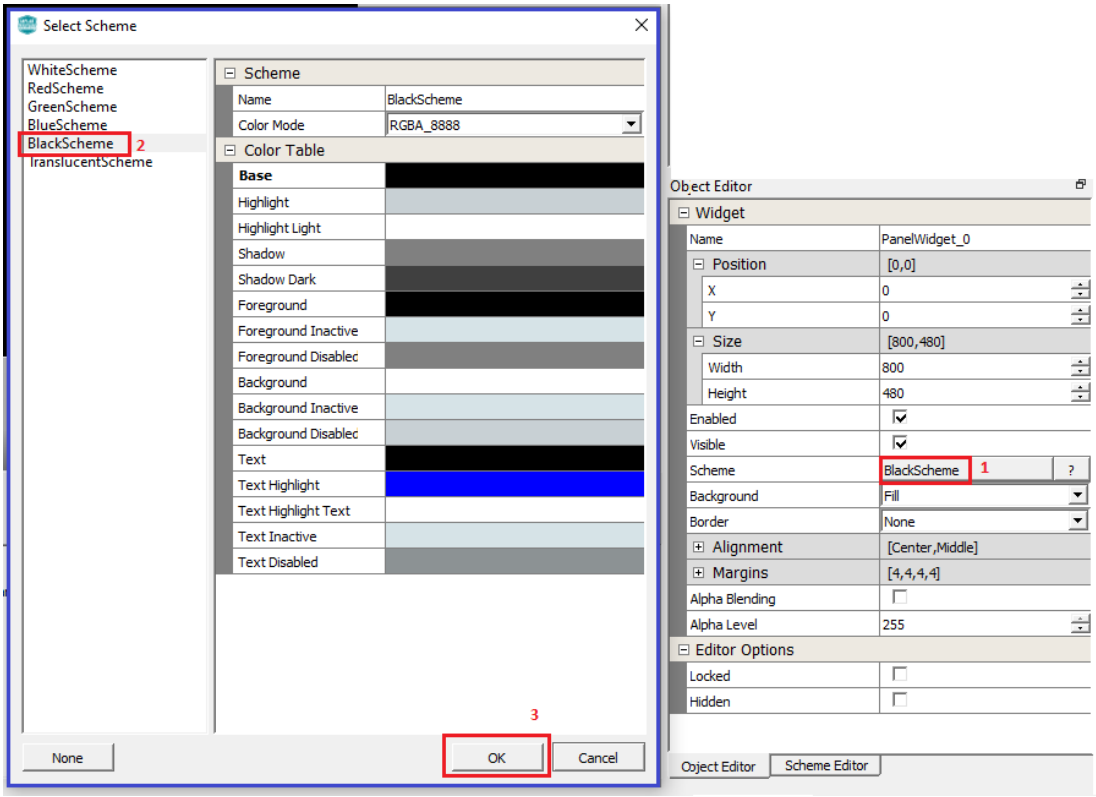

Now we are presented with a new screen to design the following:




1. In the Screen Tree manager click on the following icon to add a new layer:

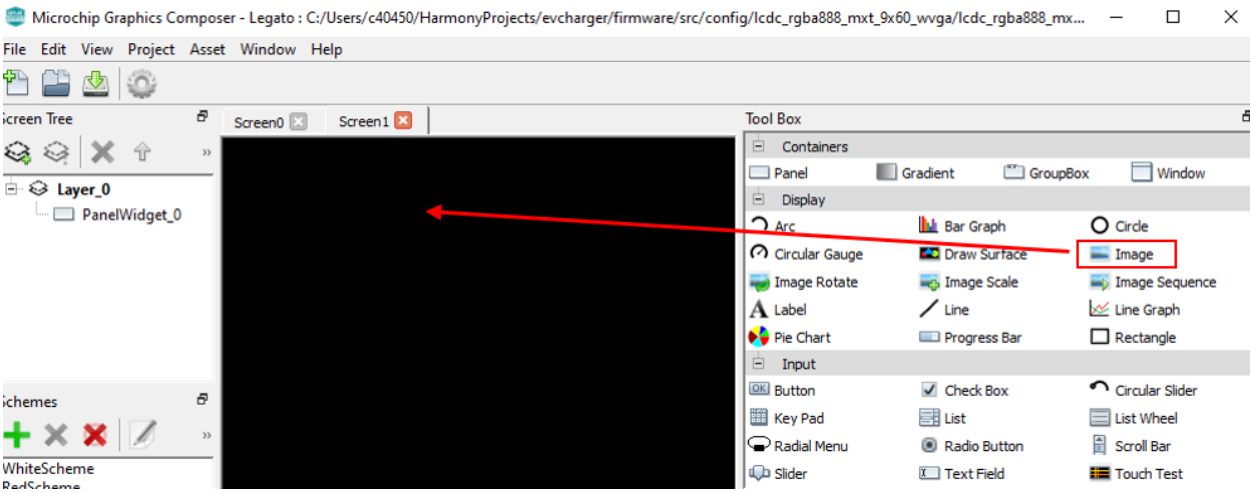

2. Drag a panel widget to the design (in Scree1) and in the “Object Editor” window, for Scheme choose “BlackScheme” that you created in Task 1.

NOTE: Click on the Scheme (as shown in #1 in figure below) and you will be presented with a “Select Scheme” window shown in blue box below. Select the required scheme (#2) and click on OK (#3).



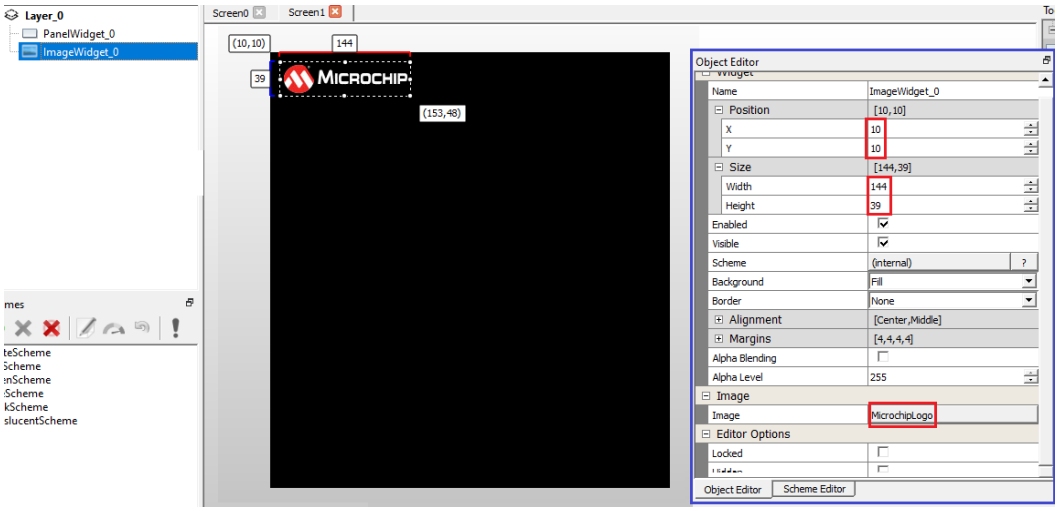

Adding Images

1. Drag an Image widget from the “Tool Box” to the screen.

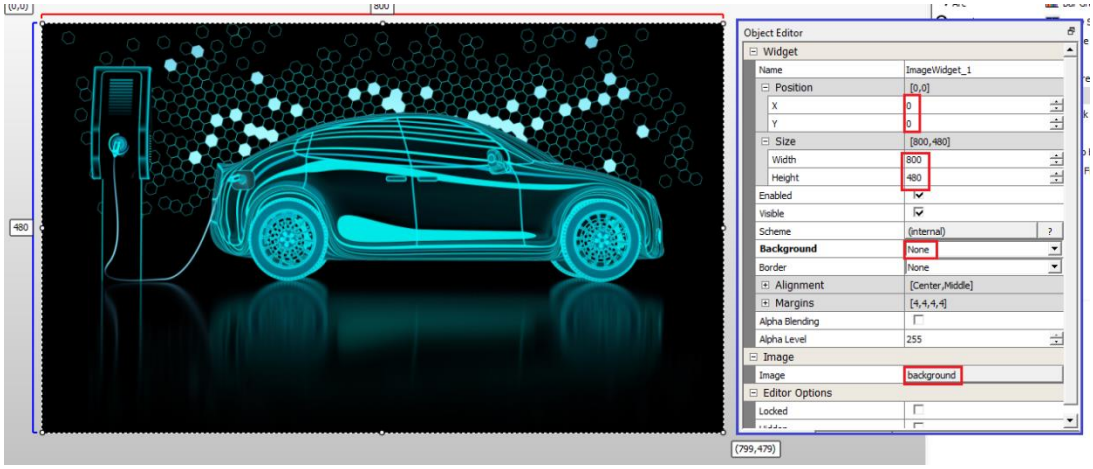



Selecting the Image widget from the screen tree, set the widget properties from the Object editor as below:

- Position – 10, 10
- Size – 144, 39
- Image – “MicrochipLogo”

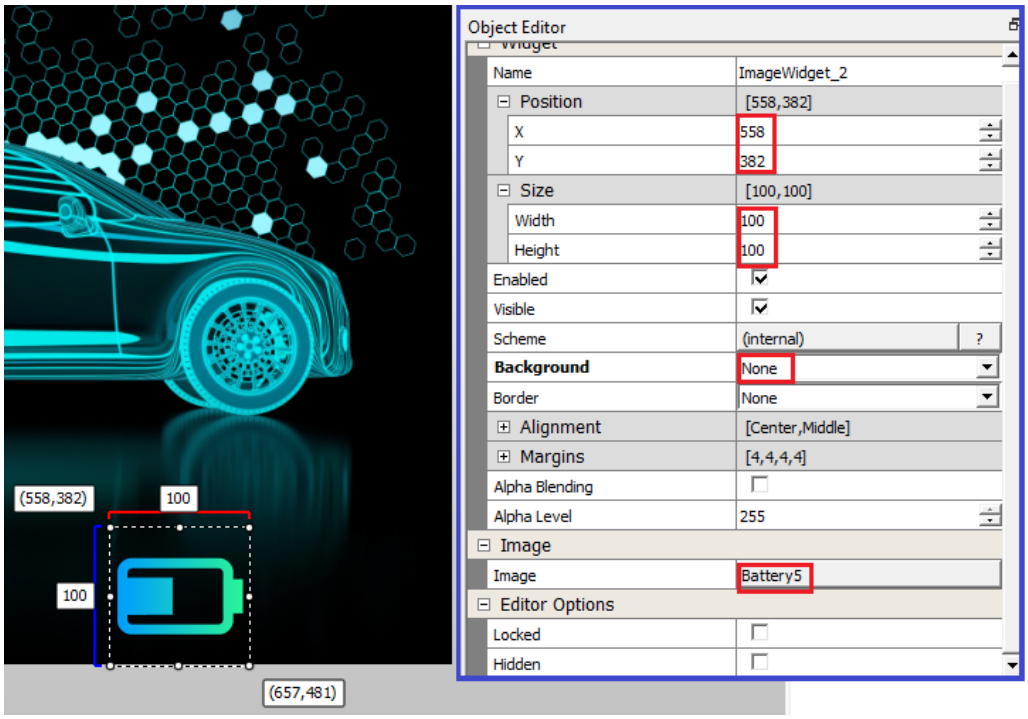



2. From the Image Manager, import “background.png” from the training resources/assets folder. Make sure the output format is set to RGB and color mode is set to RGBA_8888. Enable Compression. Please refer to Task 1 “adding Images’ section to learn how to do this.
3. Drag another “Image” widget to our design and from “Object Editor”, set the values in the blue box shown below:



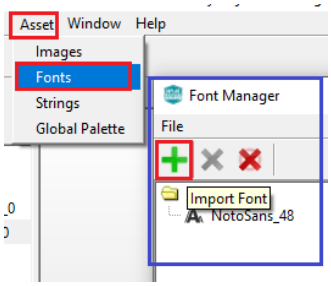

Notice how the image fills the screen and covers and blocks the “ImageWidget_0”. To make it visible, we need to push the “ImageWidget_1” behind. Select ImageWidget_1 and click on “Move Selected Objects Up” icon to ensure the “LabelWidget_0” widget is brought in front.

4. From the Image Manager, import all the BattyX.png images in the training resources/assets/sprite_assets_battery folder. Make sure their output format is set to RGB and color mode is set to RGBA_8888. Enable Compression.
5. Drag another “Image” widget to our design and from “Object Editor”, set the values in the blue box shown below:



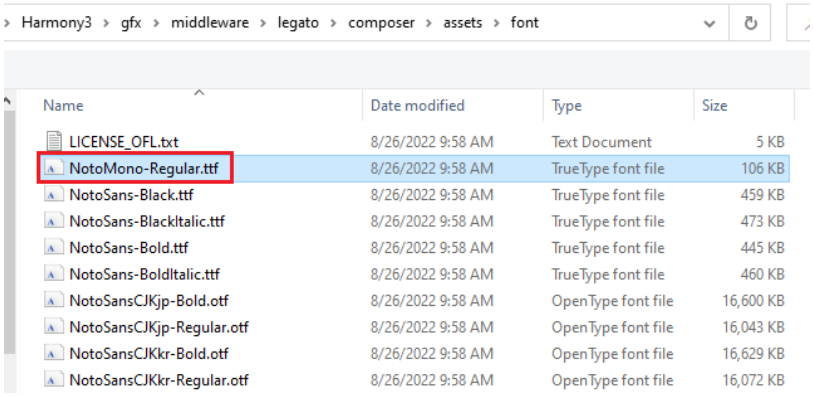

## Adding New Font

6. We need to display several strings to be displayed in a small font size, let’s say 14. We first need to add a font of size 14. To do this, click on Asset -> Fonts from the Composer main menu. From the “Font Manger” window, click the + “Import Font” icon.

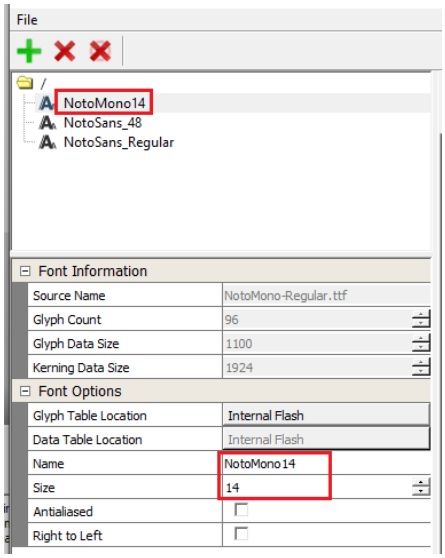



From the browse window, select NotoMono-Regular.ttf from the location – “C:\Users\c40450\harmony3_evcharger\gfx\middleware\legato\composer\assets\font”

NOTE: Please modify the highlighted path above so that it points to where you have downloaded the Harmony framework.

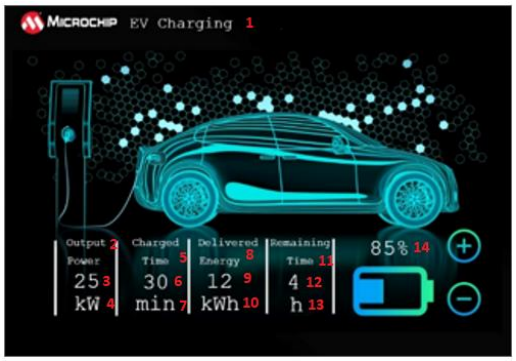



Set the font size to 14 from “Font Options” as shown below:



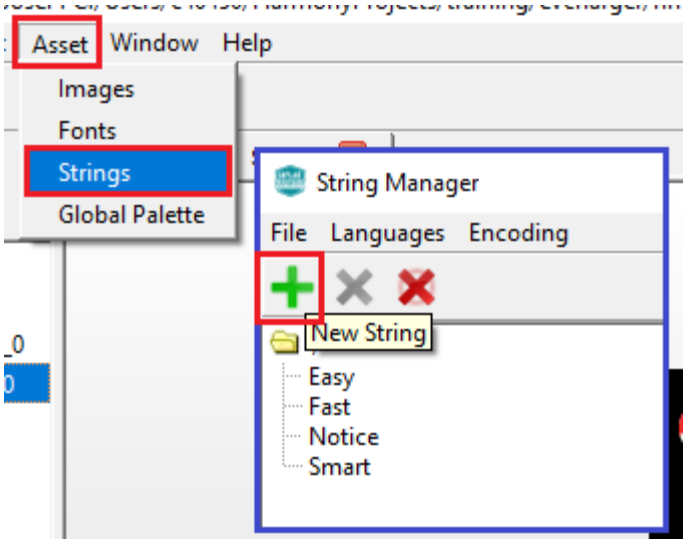

<u>Adding Strings</u>

We have 14 strings/labels in our design as shown below:

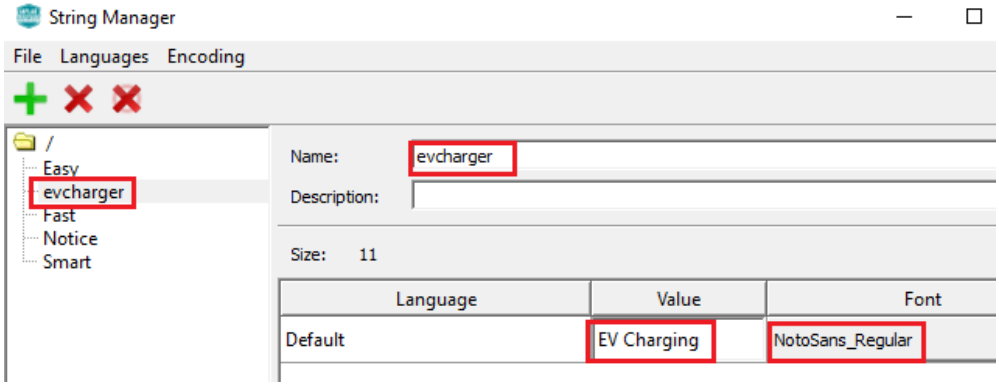



Some of these will use font size 14 that we just added and some font size 28 that is already created by the template project.

7. Click on Assets -> Strings from the Composer main menu. From the “String Manger” window, click the + “New String” icon.




Add Name(i), Value(ii) and select a Font (iii) for the String as shown below:

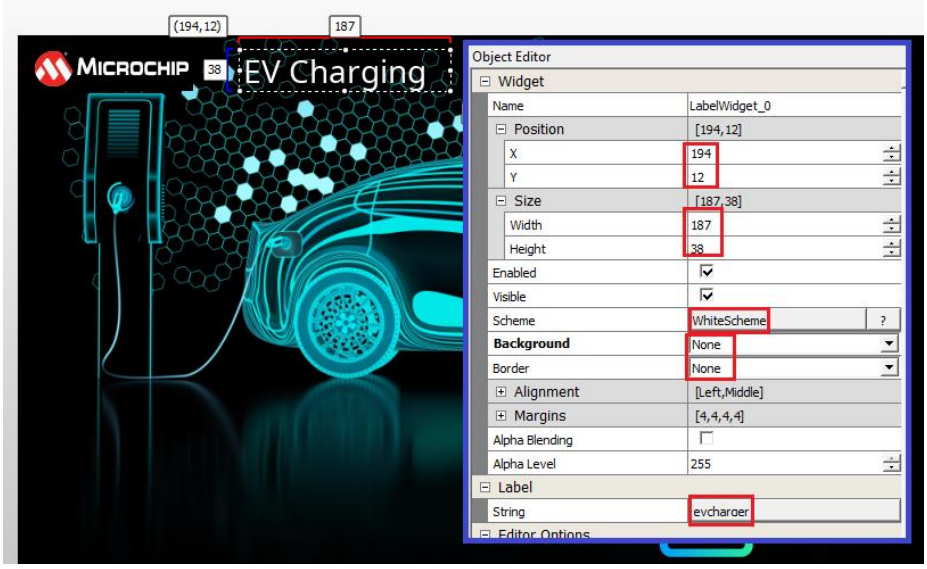



The table below shows all the strings and their settings:

| | Name | Value | Font |
|---|---|---|---|
| 1 | evcharger | EV Charging | NotoSans_Regular |
| 2 | op_label | Output\nPower | NotoMono14 |
| 3 | ctime_label | Charge\nTime | NotoMono14 |
| 4 | denergy_label | Delivered\nEnergy | NotoMono14 |
| 5 | time_label | Remaining\nTime | NotoMono14 |
| 6 | pwr | 25 | NotoSans_Regular |
| 7 | pwr_unit | kW | NotoSans_Regular |
| 8 | ctime | 30 | NotoSans_Regular |
| 9 | ctime_unit | min | NotoSans_Regular |
| 10 | energy | 12 | NotoSans_Regular |
| 11 | energy_unit | kWh | NotoSans_Regular |
| 12 | rtime | 180 | NotoSans_Regular |
| 13 | rtime_unit | min | NotoSans_Regular |
| 14 | remaining_charge | 10% | NotoSans_Regular |

Adding Labels

8. Now we are ready to create the labels for our design. Drag the “Label” widget from the “Tool Box” to the screen and from “Object Editor”, set the following values:

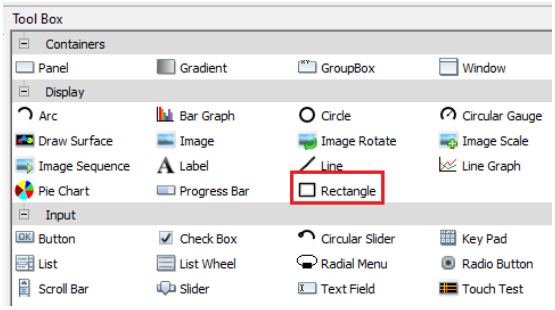



Similarly add the following Label widgets:

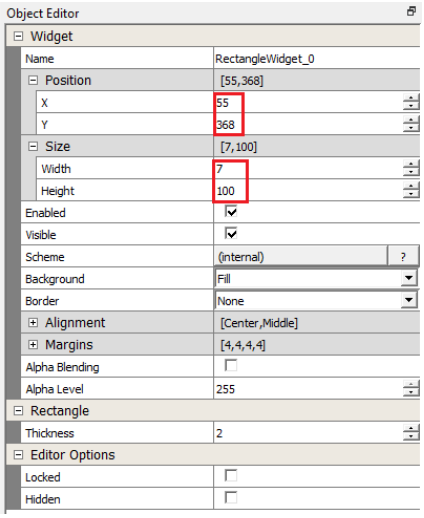
| Name | Position | | Size | | Scheme | Background | Border | Label |
|---|---|---|---|---|---|---|---|---|
| | X | Y | Width | Height | | | | |
| LabelWidget_1 | 68 | 368 | 100 | 35 | White | None | None | op_label |
| LabelWidget_2 | 178 | 368 | 100 | 35 | White | None | None | ctime_label |
| LabelWidget_3 | 275 | 368 | 100 | 35 | White | None | None | denergy_label |
| LabelWidget_4 | 390 | 368 | 100 | 35 | White | None | None | time_label |
| LabelWidget_5 | 75 | 412 | 100 | 25 | White | None | None | pwr |
| LabelWidget_6 | 75 | 447 | 100 | 25 | White | None | None | pwr_unit |
| LabelWidget_7 | 181 | 412 | 100 | 25 | White | None | None | ctime |
| LabelWidget_8 | 181 | 447 | 100 | 25 | White | None | None | ctime_unit |
| LabelWidget_9 | 275 | 412 | 100 | 25 | White | None | None | energy |
| LabelWidget_10 | 275 | 447 | 100 | 25 | White | None | None | energy_unit |
| LabelWidget_11 | 395 | 412 | 100 | 25 | White | None | None | rtime |
| LabelWidget_12 | 395 | 447 | 100 | 25 | White | None | None | rtime_unit |
| LabelWidget_13 | 580 | 364 | 100 | 25 | White | None | None | remaining_charge |

Finishing the design

9. We need 5 rectangles for our design. Drag the “Rectangle” Display widget from the Tool Box to the design:




From the Object Editor modify the Position and Size of the rectangle widget as shown in the figure below:




Drag 4 more rectangles to the design. Set the size for all of them to:

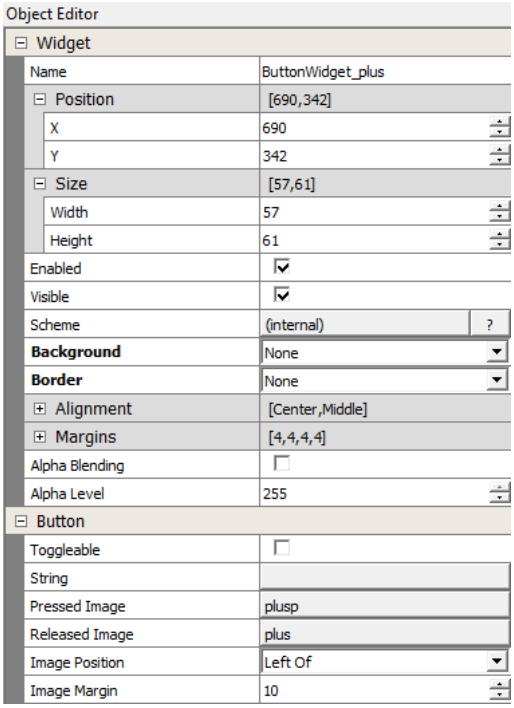
Width = 7

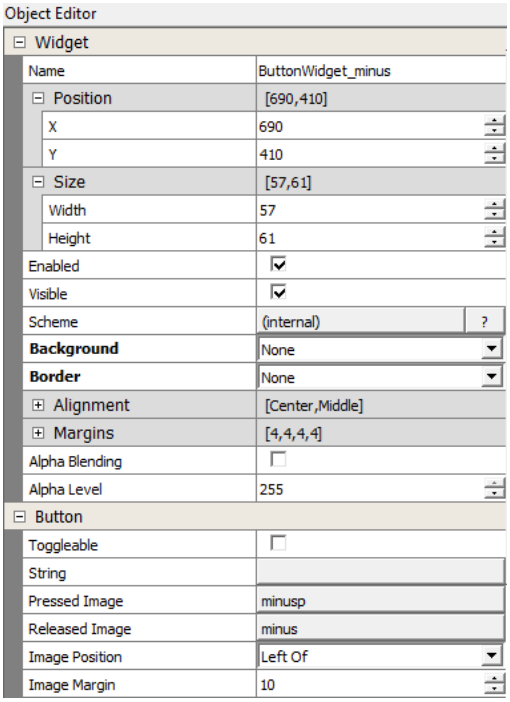
Height = 100

Change the Position of the rectangles as shown below:

| RECTANGLE | POSITION | |
|---|---|---|
| | X | Y |
| RectangleWidget_0 | 55 | 368 |
| RectangleWidget_1 | 163 | 368 |
| RectangleWidget_2 | 260 | 368 |
| RectangleWidget_3 | 378 | 368 |
| RectangleWidget_4 | 492 | 368 |

10. Next using Image Manager, import “minus.png”, “minusp.png”, “plus.png” and “plusp.png” images provided in the training resources/assets folder. Change the output format to RGB and color mode to RGBA_888. Enable compression.
11. Finally, we add the + and – buttons to our design with the following properties in the Object Editor:

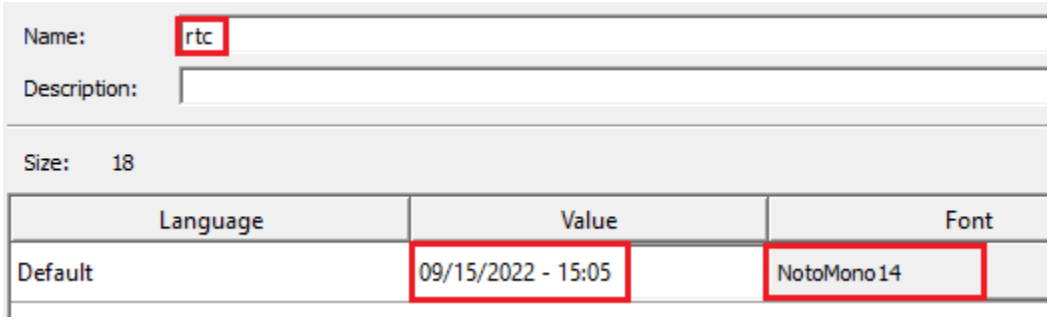


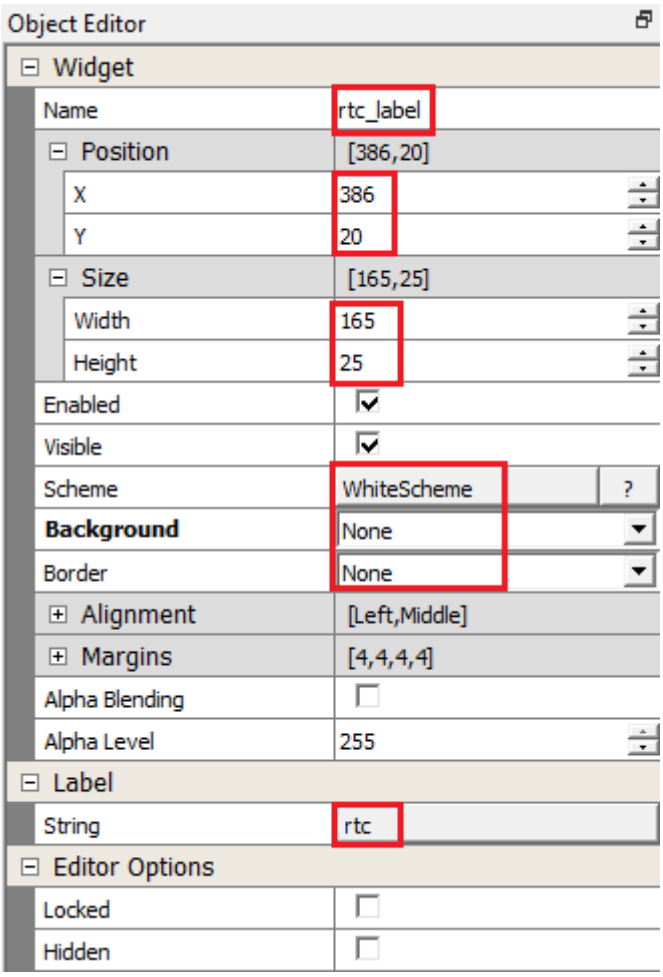

With this we have finished the design of our Screen1 for Task 2. To return to the task please click here.

# Appendix 2 – Task 3

1. Let us add a string for date / time. From Composer main menu, click Asset -> Strings. Click on + icon to add a new string as shown below:




2. In Screen1 design drag a label widget and modify the properties using Object Editor as shown below:




To return to Task 3 click here.

# Appendix 3 – Task 4

1. In app.h, add the following definitions to declare the layers and the canvas objects:

```
#define LAYER_0_CANVAS_ID 0
#define LAYER_1_CANVAS_ID 1
#define LAYER_2_CANVAS_ID 2
#define CABLE_ANIMATE_CANVAS_ID 3

#define LAYER_0_ID 0
#define LAYER_1_ID 1
#define LAYER_2_ID 2
```

2. In APP_Initialize() function in app.c, add the following lines:

```
void APP_Initialize ( void )
{
    /* Place the App state machine in its initial state. */
    appData.state = APP_STATE_INIT;


    gfxcSetLayer(LAYER_0_CANVAS_ID, LAYER_0_ID);
    gfxcSetWindowPosition(LAYER_0_CANVAS_ID, 0, 0);
    gfxcSetWindowSize(LAYER_0_CANVAS_ID, 800, 480);

    gfxcSetLayer(LAYER_1_CANVAS_ID, LAYER_1_ID);
    gfxcSetWindowPosition(LAYER_1_CANVAS_ID, 0, 0);
    gfxcSetWindowSize(LAYER_1_CANVAS_ID, 800, 480);

    gfxcSetLayer(LAYER_2_CANVAS_ID, LAYER_2_ID);
    gfxcSetWindowPosition(LAYER_2_CANVAS_ID, 0, 0);
    gfxcSetWindowSize(LAYER_2_CANVAS_ID, 800, 480);

    gfxcSetLayer(CABLE_ANIMATE_CANVAS_ID, LAYER_2_ID);
    gfxcSetWindowPosition(CABLE_ANIMATE_CANVAS_ID, 58, 48);
    gfxcSetWindowSize(CABLE_ANIMATE_CANVAS_ID, 161, 261);

    /* TODO: Initialize your application's state machine and other
     * parameters.
     */
}
```

We are setting the position and size of the 2 canvas objects and setting them to the LCD layers (0 defined in the MGC and 1) in the above lines.

3. In APP_Tasks(), APP_STATE_INIT case, replace `init_RTC_Label_Screen1` function to `init_Screen1()` (we will later implement this function in app_screen1.c):

```
/* Application's initial state. */
        case APP_STATE_INIT:
        {
            bool appInitialized = true;
            init_Screen1();
            if (appInitialized)
            {

                appData.state = APP_STATE_SERVICE_TASKS;
            }
            break;
        }
```

4. Replace the declaration of `init_RTC_Label_Screen1()` in app.c to `init_Screen1()`.

5. Add the following in app.c to include `gfx_canvas_api.h`:

```
#include "gfx/canvas/gfx_canvas_api.h"
```

6. Add the following functions in app_screen0.c - Screen0_OnShow() and Screen0_OnHide()

```
void Screen0_OnShow(void)
{
    gfxcShowCanvas(LAYER_0_CANVAS_ID);
    gfxcCanvasUpdate(LAYER_0_CANVAS_ID);
    gfxcShowCanvas(LAYER_1_CANVAS_ID);
    gfxcCanvasUpdate(LAYER_1_CANVAS_ID);
    gfxcShowCanvas(LAYER_2_CANVAS_ID);
    gfxcCanvasUpdate(LAYER_2_CANVAS_ID);
}
void Screen0_OnHide(void)
{

}
```

These lines above will display the Screen0 design to the target.

7. In app_screen1.c, add the following lines to include the header files:

```
#include "peripheral/tc/plib_tc0.h"
#include "gfx/canvas/gfx_canvas_api.h"
```

8. Next define array of leImage for the background images and a counter to cycle through these images:

```
leImage  imgAnim[14];
static uint8_t anim_cnt=0;
static uint32_t prev_tick, tick = 0;
```

9. Change the init_RTC_Label_Screen1 to **init_Screen1()** and add the line of code as highlighted below:

```
void init_Screen1(void)
{
           lastminute=0;
           leFixedString_Constructor(&p_timestring,  p_legatoTimeBuff,
      MAX_TIME_STRING_LEN *2);
           p_timestring.fn->setFont(&p_timestring, (leFont*)& NotoMono14);
           // Using a 24hr clock, with 0 based day of week and month
      (0=Sunday, 0=January)
           // 3-31-2019 23:59:50 Sunday
           currentTime.tm_hour = 23;
           currentTime.tm_min = 59;
           currentTime.tm_sec = 50;
           currentTime.tm_year = 122;
           currentTime.tm_mon = 2;
           currentTime.tm_mday = 31;
           currentTime.tm_wday = 0;
           RTC_TimeSet( &currentTime );
           init_anim_images();
           TC0_CH2_TimerCallbackRegister(TC0_CH2_TimerInterruptHandler,
      (uintptr_t)NULL);

}
```

10. Let us define the following functions in app_screen1.c:

```
void TC0_CH2_TimerInterruptHandler(TC_TIMER_STATUS status, uintptr_t
context)
{
    tick++;
}
void Screen1_OnHide(void)
{
    TC0_CH2_TimerStop();
}

void Screen1_OnShow(void)
{
    TC0_CH2_TimerStart();
    UpdateTime_Label();
    gfxcShowCanvas(LAYER_0_CANVAS_ID);
    gfxcCanvasUpdate(LAYER_0_CANVAS_ID);
    gfxcShowCanvas(LAYER_1_CANVAS_ID);
    gfxcCanvasUpdate(LAYER_1_CANVAS_ID);


}

void init_anim_images(void)
{
    imgAnim[0]= cable0;
    imgAnim[1]= cable1;
    imgAnim[2]= cable2;
    imgAnim[3]= cable3;
    imgAnim[4]= cable4;
    imgAnim[5]= cable5;
    imgAnim[6]= cable6;
    imgAnim[7]= cable7;
    imgAnim[8]= cable8;
    imgAnim[9]= cable9;
    imgAnim[10]= cable10;
    imgAnim[11]= cable11;
    imgAnim[12]= cable12;
    imgAnim[13]= cable13;
}
```

In init_screen1(), we register a timer callback function - TC0_CH2_TimerInterruptHandler. In this interrupt handler we increment a tick count. In init_anim_images() function, we create an array of (leImage) images with the cable animation. The Screen1_OnShow() function starts the timer and updates the canvas objects.

11. Add a function **Update_Screen1()** and call UpdateTime_Label() function you wrote in the previous task. Add a function to update the cable animation images and call this function in the Update_Screen1() function:

```
void UpdateChargeAnime_Canvas(void)
{
       if (tick != prev_tick)
           {
               prev_tick = tick;
               gfxcSetPixelBuffer(CABLE_ANIMATE_CANVAS_ID,
                                  161,
                                  261,
                                  GFX_COLOR_MODE_RGBA_8888,
                                  (void *) imgAnim[++anim_cnt].buffer.pixels);
```

```
            if(anim_cnt >=13)
                anim_cnt =0;

            gfxcShowCanvas(LAYER_1_CANVAS_ID);
            gfxcCanvasUpdate(LAYER_1_CANVAS_ID);
            gfxcShowCanvas(CABLE_ANIMATE_CANVAS_ID);
            gfxcCanvasUpdate(CABLE_ANIMATE_CANVAS_ID);

    }
}

void Update_Screen1(void)
{
    UpdateTime_Label();
    UpdateChargeAnime_Canvas();
}
```

12. Declare Update_Screen1() in app.h and call the function in APP_Tasks() in app.c:

```
case APP_STATE_SERVICE_TASKS:
        {
            if(legato_getCurrentScreen()==screenID_Screen1)
                Update_Screen1();

            break;
        }
```

13. In UpdateTime_Label(), we need to update the canvas layer assigned to layer0 since RTC label is in layer 0. To do this, call the gfxcCanvasUpdate()as shown in the figure below:

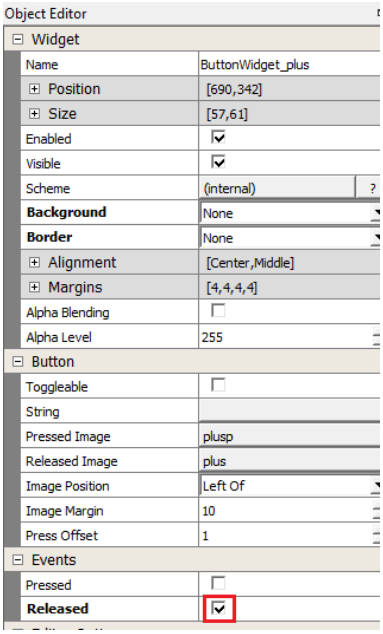
```
       void UpdateTime_Label (void)
       {
           RTC_TimeGet( &currentTime );
           if(lastminute != currentTime.tm_min)
           {
               lastminute = currentTime.tm_min;
               UpdateTime();
               p_timestring.fn->setFromCStr(&p_timestring, p_timecharbuff);

               Screen1_rtc_label->fn->setString(Screen1_rtc_label,
       (leString*)&p_timestring);
               gfxcShowCanvas(LAYER_1_CANVAS_ID);
               gfxcCanvasUpdate(LAYER_1_CANVAS_ID);
           }
}
```

To return to Task 4 click here.

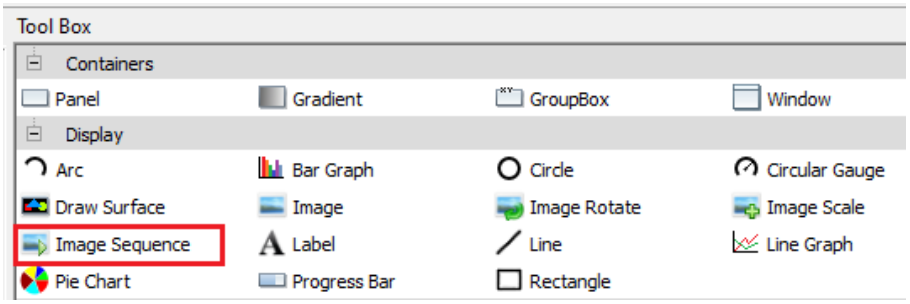
# Appendix 4 – Task 5

Changes to Composer

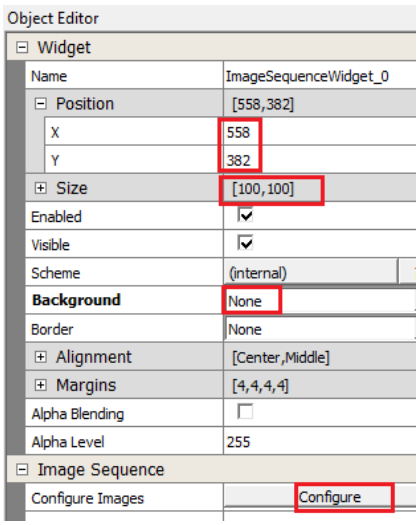
1. Click the “ButtonWidget_plus” widget and from the Object Editor select “Released” checkbox as shown below:

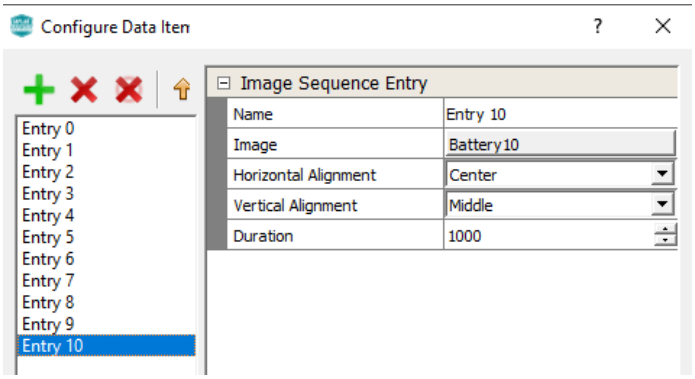



   Select the Release button event for ButtonWidget_minus widget as well.
2. We already added the 11 images “BatteryX.png” available in the “sprite_assets_battery” folder in a previous task.
3. Select ImageWidget_2 icon from Screen1, Layer_0 and remove the widget using the “Delete Selected Objects” icon ✖.
4. Select Image Sequence widget from the Tool Box and drag it to the Layer_1 of Screen1 in your design:




5. From the Object editor modify the properties of the “ImageSequenceWidget_0” Image Sequence widget as shown below. Click on Configure button and select the 11 battery images you just added.




6. Regenerate code using MCC tool.

Return to Task 5 by clicking here.

# Appendix 5 – Task 5

Adding Application Code

The ⊕ and ⊖ buttons will change the "Battery Charge %", "Remaining Time" and "Delivered Energy". To calculate these values let us use the following assumptions:

i. OP (Output Power) = 25KW (*fixed*)
ii. BS (Battery Size) = 50 KWh (*fixed*)
iii. P (% Battery Charged) (*default value 10%*)
iv. RT (Remaining Time) = (BS/OP) * (100 -P) min
v. CT (Charged Time) = P min (1 min per % charge)
vi. DE (Delivered Energy) = OP x CT KWh

1. In app_screen1.c, let us define the following:

```
#define OUTPUT_POWER 25 //unit KW
#define BATTERY_SIZE 50  //unit KWh
uint8_t percent_battery_charge = 10;
int remaining_time = 0;  // unit min
int charged_time = 0;  //unit min
int delivered_energy  = 0; // unit KWh

leFixedString p_percentage;
leFixedString p_remainingT;
leFixedString p_chargeT;
leFixedString p_deliveredE;

static leChar p_legato_percentageBuff[5] = {0};
static leChar p_legato_remainingT_Buff[4] = {0};
static leChar p_legato_chargeT_Buff[4] = {0};
static leChar p_legato_deliveredE_Buff[4] = {0};

static char p_percentcharbuff[5];
static char p_remainingtimecharbuff[4];
static char p_chargeTcharbuff[4];
static char p_deliveredEcharbuff[4];
```

2. With the Graphics Composer code regeneration done previously, you should see "event_Screen1_ButtonWidget_plus_OnReleased" and "event_Screen1_ButtonWidget_minus_OnReleased" declared in le_gen_screen_Screen1.h. Let us define these functions in app_Screen1:

```
void event_Screen1_ButtonWidget_plus_OnReleased(leButtonWidget* btn)
{
    if(percent_battery_charge>=100){
        percent_battery_charge=100;
    }else{
        percent_battery_charge+=10;
        calculate_change();
        update_meterLabels();
        Screen1_ImageSequenceWidget_0->fn-
>showNextImage(Screen1_ImageSequenceWidget_0);
    }
```

```
        }

        void event_Screen1_ButtonWidget_minus_OnReleased(leButtonWidget* btn)
        {
            if(percent_battery_charge<=0){
                percent_battery_charge=0;
            }else{
                percent_battery_charge-=10;
                calculate_change();
                update_meterLabels();
                Screen1_ImageSequenceWidget_0->fn-
        >showPreviousImage(Screen1_ImageSequenceWidget_0);
            }
}
```

When the ⊕ button is pressed, the “Percentage Charge” is incremented in 10s and the charge time increases with it. Similarly when the ⊖ button is pressed, the “Percentage Charge” is decremented in 10s and the charge time decreases with it. The `calculate_change()` function calculates “remaining time”, “charge time” and “delivered energy”. The `update_label()` function updates the leFixedString and leLabelWidget of the assets highlighted below:


The image assigned to `Screen1_ImageSequenceWidget_0` is changed using `showNextImage` or `showPreviousImage` API in the button event handler function.

3. The `calculate_change()` function and `update_label()` function are defined as below:

```
        void calculate_change(void)
        {
            remaining_time = (BATTERY_SIZE/OUTPUT_POWER)*(100-
        percent_battery_charge);
            charged_time = percent_battery_charge;
            delivered_energy = (OUTPUT_POWER * charged_time)/60;
        }

        void update_meterLabels(void)
        {
            memset(p_percentcharbuff,0,sizeof(p_percentcharbuff));
            sprintf(p_percentcharbuff,"%d%%",percent_battery_charge);
            p_percentage.fn->setFromCStr(&p_percentage, p_percentcharbuff);
            Screen1_LabelWidget_13->fn->setString(Screen1_LabelWidget_13,
        (leString*)&p_percentage);

            memset(p_remainingtimecharbuff,0,sizeof(p_remainingtimecharbuff));
            sprintf(p_remainingtimecharbuff,"%d",remaining_time);
            p_remainingT.fn->setFromCStr(&p_remainingT, p_remainingtimecharbuff);
            Screen1_LabelWidget_11->fn->setString(Screen1_LabelWidget_11,
        (leString*)&p_remainingT);

            memset(p_chargeTcharbuff,0,sizeof(p_chargeTcharbuff));
            sprintf(p_chargeTcharbuff,"%d",charged_time);
            p_chargeT.fn->setFromCStr(&p_chargeT, p_chargeTcharbuff);
            Screen1_LabelWidget_7->fn->setString(Screen1_LabelWidget_7,
        (leString*)&p_chargeT);
```

```
            memset(p_deliveredEcharbuff,0,sizeof(p_deliveredEcharbuff));
            sprintf(p_deliveredEcharbuff,"%d",delivered_energy);
            p_deliveredE.fn->setFromCStr(&p_deliveredE, p_deliveredEcharbuff);
            Screen1_LabelWidget_9->fn->setString(Screen1_LabelWidget_9,
        (leString*)&p_deliveredE);

            gfxcShowCanvas(LAYER_1_CANVAS_ID);
            gfxcCanvasUpdate(LAYER_1_CANVAS_ID);
}
```

4. Add the following in init_Screen1():

```
leFixedString_Constructor(&p_percentage,  p_legato_percentageBuff, 8);
p_percentage.fn->setFont(&p_percentage, (leFont*)&NotoSans_Regular);

leFixedString_Constructor(&p_remainingT,  p_legato_remainingT_Buff, 8);
p_remainingT.fn->setFont(&p_remainingT, (leFont*)&NotoSans_Regular);

leFixedString_Constructor(&p_chargeT,  p_legato_chargeT_Buff, 8);
p_chargeT.fn->setFont(&p_chargeT, (leFont*)&NotoSans_Regular);

leFixedString_Constructor(&p_deliveredE,  p_legato_deliveredE_Buff, 8);
p_deliveredE.fn->setFont(&p_deliveredE, (leFont*)&NotoSans_Regular);
```

5. In `Screen1_OnShow()` function add the following:

```
Screen1_ImageSequenceWidget_0->fn->showImage(Screen1_ImageSequenceWidget_0,1);
```

To return to Task 5, click here.

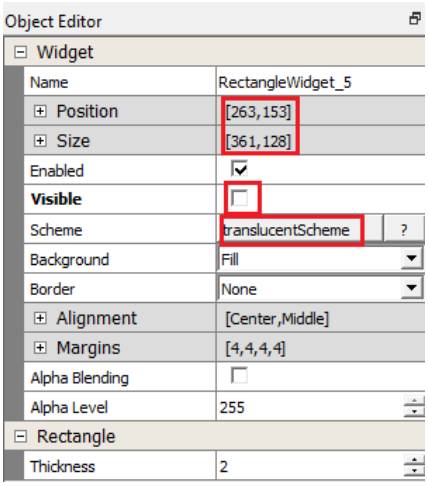
# Appendix 6 – Task 6

Changes to design using Legato Graphics Composer

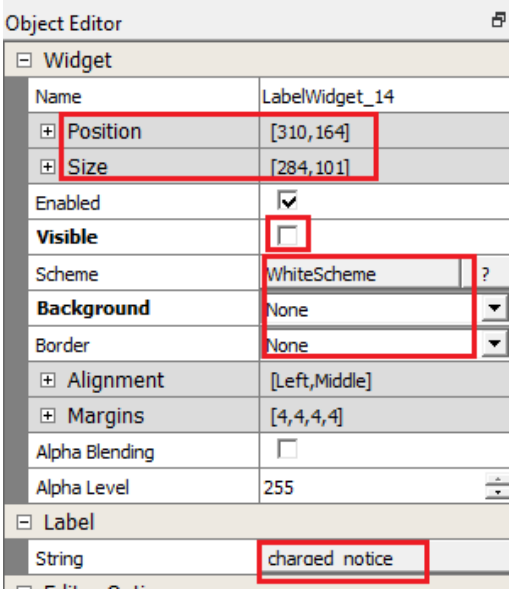
1. From String Manager, add the following strings:

| Name | Value | Font |
|---|---|---|
| charging_notice | Charging . . . | NotoSans_48 |
| charged_notice | Battery fully \ncharged | NotoSans_48 |
| charge_percent | | NotoSans_Regular |

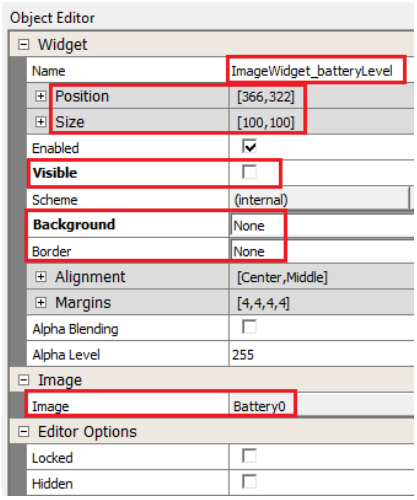
2. Select Screen1 tab. Drag Rectangle widget to your design in layer 1 and set its properties as shown below:



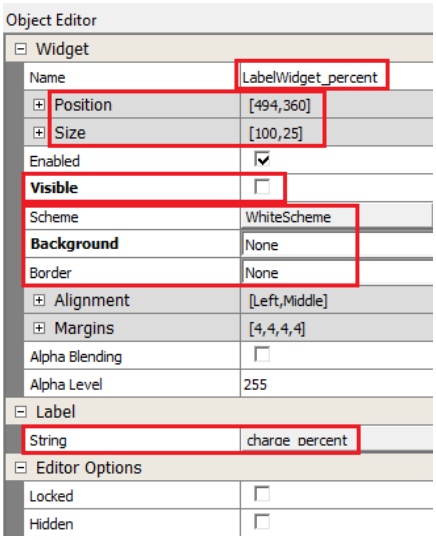

3. Drag label widget to layer 1 of Screen1 and set its properties as shown below:



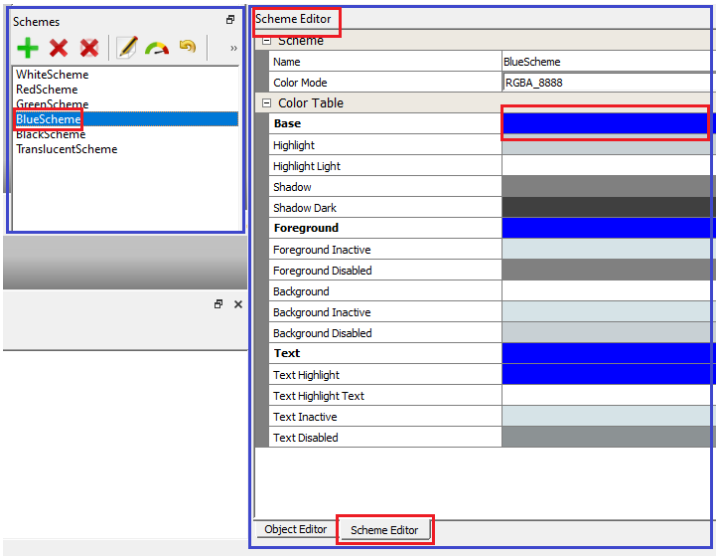

4. Select Screen0 tab. Drag an Image widget from the tool box to your design. Set its properties using Object Editor as shown below:



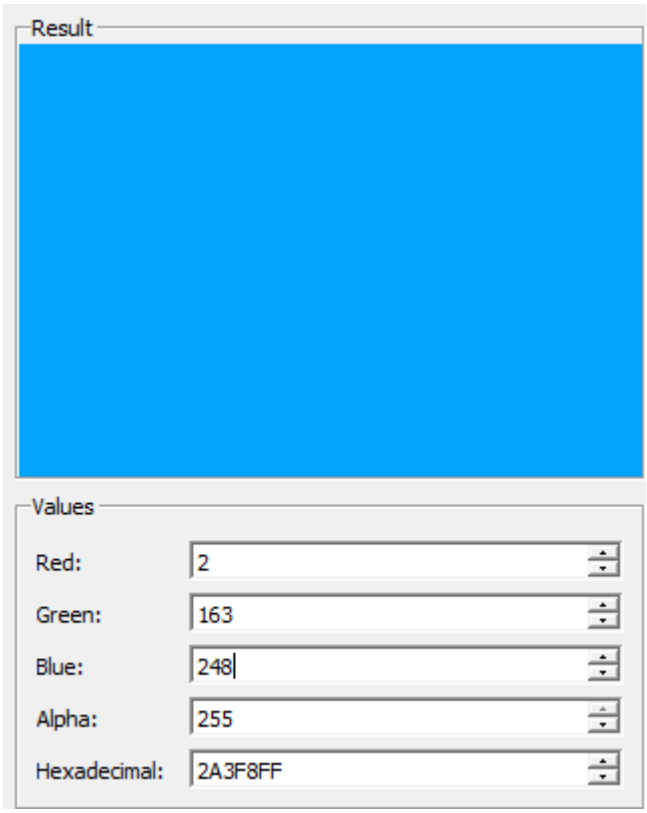

5. Drag Label widget from tool box to Screen0 design and set its properties using Object Editor as shown below:



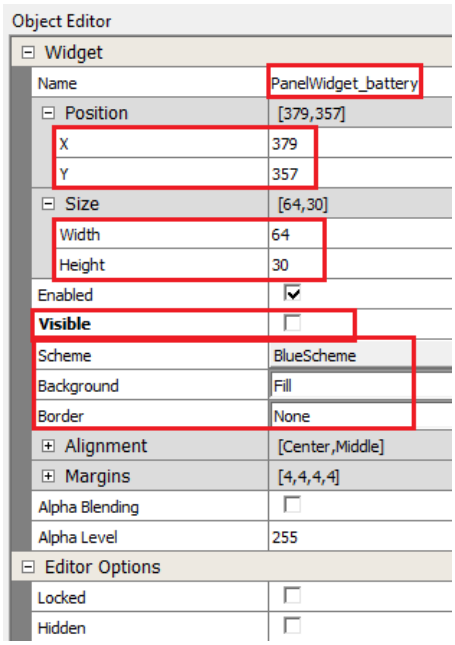

6. Let us modify the blue scheme for use with the panel widget we will add next. From Schemes tab on the bottom left, select BlueScheme and from the Scheme Editor tab on the right, click on Base color.




7. In the color editor, enter the following RGB values and click OK:




8. Drag Panel widget from the tool box (available in the Containers category) to the Screen0 design and modify its properties in the Object Editor as shown below:




To return to task 6, click here.

# Appendix 7 – Task 6

Adding Application Code

1. In app.h, let us define an enum SCREEN_STATE to keep track of which screen we are in and add this to APP_DATA as shown below:

```
typedef enum
{
    DEMO_SCREEN0=0,
    DEMO_SCREEN1,
    DEMO_SCREEN2,
    DEMO_SCREEN3,

} SCREEN_STATES;
typedef struct
{
    /* The application's current state */
    APP_STATES state;
    SCREEN_STATES demo_screen;
    /* TODO: Define any additional data used by the application. */

} APP_DATA;
```

In app.h let us also declare the following functions and variables:

```
extern uint8_t percent_battery_charge;
extern int     remaining_time;  //unit min
extern int     charged_time;   // unit min
extern int     delivered_energy; // unit KWh
void init_Screen0(void);
void Update_Screen2(void);

SCREEN_STATES getScreen(void);
void setScreen(SCREEN_STATES screen);
uint8_t get_screensaver_cnt(void);
```

The getScreen and setScreen functions are defined in app.c. The get_screensaver_cnt function is implemented in app_screen1.c.

2. In app_scene1.c, in init_Screen1(), add the following code to register a TC0_CH1 Timer callback function:

```
    TC0_CH1_TimerCallbackRegister(TC0_CH1_TimerInterruptHandler,
(uintptr_t)NULL);
```

3. Add the timer interrupt handler `TC0_CH1_TimerInterruptHandler`:

```
uint8_t screensaver_cnt = 0;
void TC0_CH1_TimerInterruptHandler(TC_TIMER_STATUS status, uintptr_t context)
{
    screensaver_cnt++;
}
```

NOTE: If the ⊕ and ⊖ buttons are pressed the `screensaver_cnt` should be reset to 0 in event_Screen1_ButtonWidget_plus_OnReleased() and event_Screen1_ButtonWidget_minus_OnReleased():

```
void event_Screen1_ButtonWidget_plus_OnReleased(leButtonWidget* btn)
{
    screensaver_cnt = 0;
    [..]
}
void event_Screen1_ButtonWidget_minus_OnReleased(leButtonWidget* btn)
{
    screensaver_cnt =0;
    [..]
}
```

4. Update Screen1_OnShow as shown below:

```
    void Screen1_OnShow(void)
    {

        if(getScreen()== DEMO_SCREEN0)
        {
            setScreen(DEMO_SCREEN1);
            TC0_CH1_TimerStart();
            TC0_CH2_TimerStart();
            Screen1_ImageSequenceWidget_0->fn-
    >showImage(Screen1_ImageSequenceWidget_0,1);
        }else if(getScreen()== DEMO_SCREEN2)
        {
            setScreen(DEMO_SCREEN3);
            remaining_time = 0;
            charged_time = percent_battery_charge;
            delivered_energy = (OUTPUT_POWER * charged_time)/60;
            update_meterLabels();
            Screen1_RectangleWidget_5->fn->setVisible(Screen1_RectangleWidget_5,
    LE_TRUE);
            Screen1_LabelWidget_14->fn->setVisible(Screen1_LabelWidget_14,
    LE_TRUE);
            Screen1_ImageSequenceWidget_0->fn-
    >showImage(Screen1_ImageSequenceWidget_0,(percent_battery_charge/10));
        }

        gfxcShowCanvas(LAYER_0_CANVAS_ID);
        gfxcCanvasUpdate(LAYER_0_CANVAS_ID);
        gfxcShowCanvas(LAYER_1_CANVAS_ID);
        gfxcCanvasUpdate(LAYER_1_CANVAS_ID);

    }
```

5. In Screen1_OnHide() we stop both timers:

```
void Screen1_OnHide(void)
{
    TC0_CH2_TimerStop();
    TC0_CH1_TimerStop();
}
```

6. Finally, let us add a function that returns the screensaver_cnt which will decide when to transition to Screen0. If screensaver_cnt >=2, then we show the following screen (Screen0):

```
uint8_t get_screensaver_cnt(void)
{
    return screensaver_cnt;
}
```

7. In app.c, let us define a function that sets the SCREEN_STATE we are in:

```
SCREEN_STATES getScreen(void)
{
    return appData.demo_screen;
}
void setScreen(SCREEN_STATES screen)
{
    appData.demo_screen = screen;
}
```

8. In APP_Initialize() function, we initialize the SCREEN_STATE to SCREEN0:

```
void APP_Initialize ( void )
{
    /* Place the App state machine in its initial state. */
    appData.state = APP_STATE_INIT;
    appData.demo_screen = DEMO_SCREEN0;
    [..]
```

9. In APP_Tasks(), case APP_STATE_INIT we add init_Screen0 () function call as shown below:

```
case APP_STATE_INIT:
        {
            bool appInitialized = true;

            init_Screen1();
            init_Screen0();
            if (appInitialized)
            {

                appData.state = APP_STATE_SERVICE_TASKS;
            }
            break;
```

10. We modify APP_Tasks(), case APP_STATE_SERVICE_TASKS as shown below:

```
case APP_STATE_SERVICE_TASKS:
        {
            if(appData.demo_screen == DEMO_SCREEN1){
                Update_Screen1();
                if(get_screensaver_cnt() >=2)
                    legato_showScreen(screenID_Screen0);
            }
            if(appData.demo_screen == DEMO_SCREEN2){
                Update_Screen2();
            }

            break;
        }
```

11. In app_screen0.c, we need to register a callback function for TC1_CH0. Add a function called init_Screen0() as shown below:

```
void init_Screen0(void)
{
    TC1_CH0_TimerCallbackRegister(TC1_CH0_TimerInterruptHandler,
(uintptr_t)NULL);
    leFixedString_Constructor(&p_BattPercentage,  p_BattPercentageBuff, 8);
```

```
    p_BattPercentage.fn->setFont(&p_BattPercentage, (leFont*)&
NotoSans_Regular);
}
```

Declare the following variables globally to dynamically change the percentage of battery charge:

```
leFixedString p_BattPercentage;
static leChar p_BattPercentageBuff[5] = {0};
static char p_BatPercentcharbuff[5];
```

12. In Screen0_OnShow function in app_screen0.c, we need to check if the application is in SCREEN_STATES.DEMO_SCREEN2. If so, we will do the following:
    i. Disable ButtonWidget_0 and make it invisible
    ii. Change string assigned to LabelWidget_0 from “Notice” to “charging_notice”
    iii. Make ImageWidget_batteryLevel visible
    iv. Make LabelWidget_percent visible and update its value based on the value selected in Screen1
    v. Start the TC1CH0 to animate the battery level image.

    To do these, add the following code in Screen0_OnShow():

```
void updateBattPercentage(void)
{
    memset(p_BatPercentcharbuff,0,sizeof(p_BatPercentcharbuff));
    sprintf(p_BatPercentcharbuff,"%d%%",percent_battery_charge);
    p_BattPercentage.fn->setFromCStr(&p_BattPercentage, p_BatPercentcharbuff);
    Screen0_LabelWidget_percent->fn->setString(Screen0_LabelWidget_percent,
(leString*)&p_BattPercentage);
}

void Screen0_OnShow(void)
{
    if(getScreen() == DEMO_SCREEN1)
    {
        setScreen(DEMO_SCREEN2);
        Screen0_ButtonWidget_0->fn->setVisible(Screen0_ButtonWidget_0, LE_FALSE);
        Screen0_ButtonWidget_0->fn->setEnabled(Screen0_ButtonWidget_0, LE_FALSE);
        Screen0_LabelWidget_0->fn->setString(Screen0_LabelWidget_0,
(leString*)&string_charging_notice);
        Screen0_ImageWidget_batteryLevel->fn->setVisible(Screen0_ImageWidget_batteryLevel, LE_TRUE);
        updateBattPercentage();
        Screen0_PanelWidget_battery->fn->setWidth(Screen0_PanelWidget_battery, (64*0.1));
        Screen0_LabelWidget_percent->fn->setVisible(Screen0_LabelWidget_percent, LE_TRUE);
        Screen0_PanelWidget_battery->fn->setVisible(Screen0_PanelWidget_battery, LE_TRUE);
        TC1_CH0_TimerStart();
    }


    gfxcShowCanvas(LAYER_0_CANVAS_ID);
    gfxcCanvasUpdate(LAYER_0_CANVAS_ID);
    gfxcShowCanvas(LAYER_1_CANVAS_ID);
    gfxcCanvasUpdate(LAYER_1_CANVAS_ID);
    gfxcShowCanvas(LAYER_2_CANVAS_ID);
    gfxcCanvasUpdate(LAYER_2_CANVAS_ID);
}
```

13. We need the TC1CH0 timer to update a tick count every millisecond to animate the battery level and also another tick to keep track of 10 seconds. After 10 seconds, Screen1 is displayed with the “Battery Charged” message. Therefore, add the following timer interrupt handler in app_screen0.c:

```
static uint32_t prev_batt_tick, battery_tick=0;
int sec_cntr=0;
void TC1_CH0_TimerInterruptHandler(TC_TIMER_STATUS status, uintptr_t context)
{
    battery_tick++;
    if(battery_tick>=10){
        sec_cntr++;
        battery_tick=0;
    }
}
```

14. We now write the following function in app_screen0.c to animate the battery for 10 seconds after which we display Screen1.

```
    int i =0;

    void Update_Screen2(void)
    {
        if (battery_tick != prev_batt_tick)
        {
            prev_batt_tick = battery_tick;

            if(i<=(percent_battery_charge/10)){
                /*Screen0_PanelWidget_battery->fn-
    >setWidth(Screen0_PanelWidget_battery, (64*0.1*++i));*/
                gfxcShowCanvas(LAYER_2_CANVAS_ID);
                gfxcCanvasUpdate(LAYER_2_CANVAS_ID);
            }else{
                i=0;
            }
        }
        if(sec_cntr >= 10){
            TC1_CH0_TimerStop();
            legato_showScreen(screenID_Screen1);
        }
    }
```

To return to Task 6, click here.

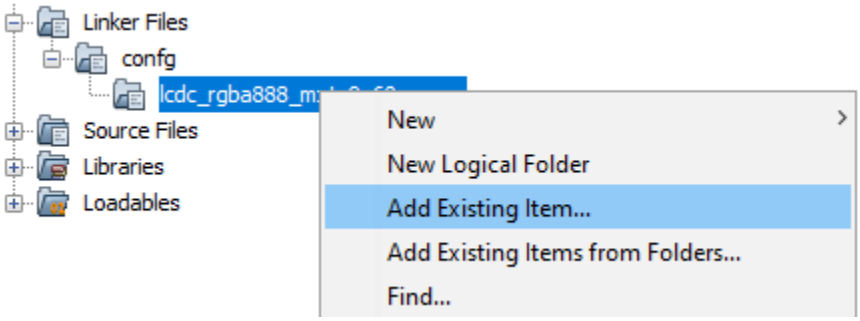
# Common Trouble Shooting Issues

1. After performing MCC Code Regeneration, the linker file does not get included in the project
   **Reason:** The linker file should be added in the following section in the evcharger_9x60_cu_wvga.X\nbproject\configuration.xml:

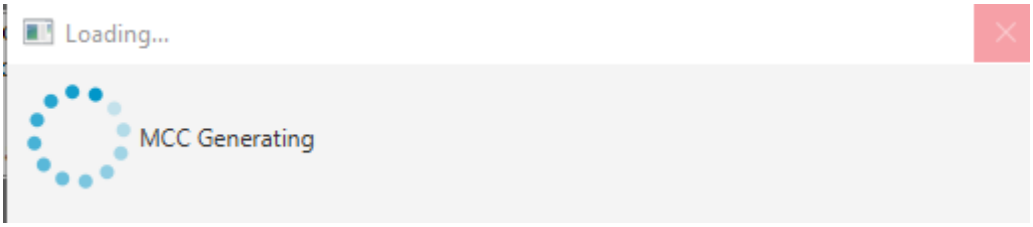
```
<logicalFolder name="LinkerScript"
               displayName="Linker Files"
               projectFiles="true">
    <logicalFolder name="config" displayName="config" projectFiles="true">
     <logicalFolder name="lcdc_rgba888_mxt_9x60_wvga"
                    displayName="lcdc_rgba888_mxt_9x60_wvga"
                    projectFiles="true">
      <itemPath>../src/config/lcdc_rgba888_mxt_9x60_wvga/ddram.ld</itemPath>
     </logicalFolder>
    </logicalFolder>
   </logicalFolder>
```

   Instead it gets added to the rest of the source files.

   **Fix:** Manually delete the following line from configuration.xml:
   <itemPath>../src/config/lcdc_rgba888_mxt_9x60_wvga/ddram.ld</itemPath>
   In the Project explorer, right click on the "Linker Files" folder (you can add logical folders -> config/ lcdc_rgba888_mxt_9x60_wvga), and click "Add Existing Item". Manually add ddram.ld generated in the "<path>\evcharger\firmware\src\config\lcdc_rgba888_mxt_9x60_wvga" folder.




2. The "MCC Generating" window doesn't seem to go away even though it has been running for a really long time and seems like the project folders have been generated.




   **Reason:** The following error has occurred (check the log file in C:\Users\c40450\.mcc\log):

```
java.lang.RuntimeException: java.lang.reflect.InvocationTargetException
        at javafx.fxml.FXMLLoader$MethodHandler.invoke(FXMLLoader.java:1774)
```

**Fix:** Kill the MPLAB X IDE using task manager and relaunch.

3. You get the following compiler error:
   make[2]: *** No rule to make target '../src/config/lcdc_rgba888_mxt_9x60_wvga/gfx/legato/widget/circulargauge/legato_widget_circular_gauge_skin_classic.c', needed by 'build/lcdc_rgba888_mxt_9x60_wvga/debug/_ext/2122148963/legato_widget_circular_gauge_skin_classic.o'. Stop.
   make[2]: *** Waiting for unfinished jobs....
   **Reason:** The toolchain is not able to find the file even though it exists in the project.
   **Fix:** Since this file is not used in our design, right click on the file in the project explorer and select “Exclude file(s) from current configuration”.