Task 1 – Managing simple widgets: Image, Button, Labels, Schemes

Hardware Connection

<Explain how to connect hardware with picture>

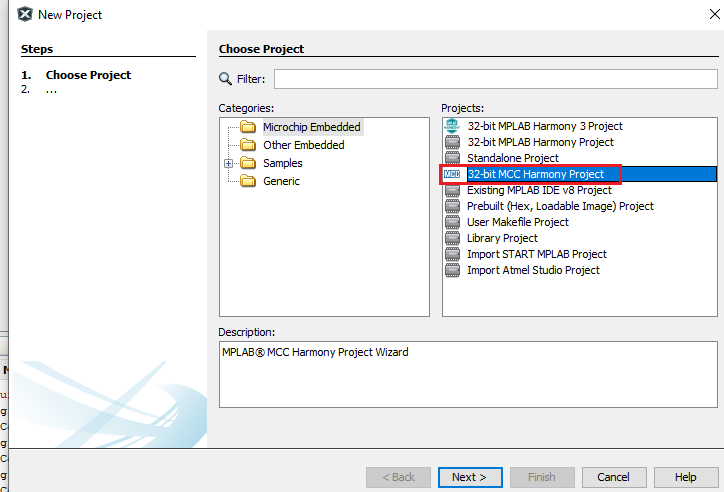
NOTE: Please ensure that the Black GND wire is connected to pin 1 (GND) of J11 of the target board.

Pic of Target board + power+ debugger

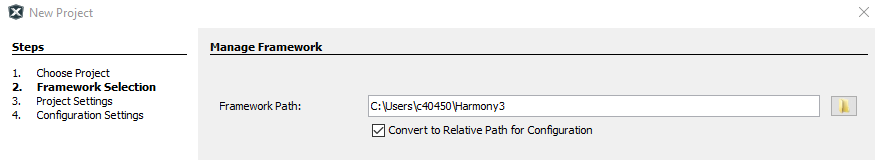
+ Debugger cable

Creating New Harmony Project

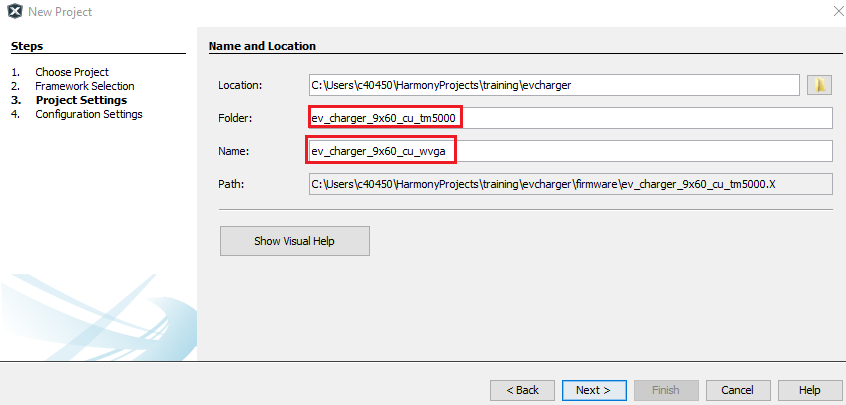
1. Click on File -> New Project. Please make sure you choose “32-bit CC Harmony Project”:



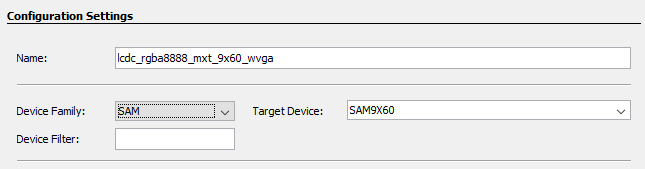
1. For Framework Path, choose the location where Harmony repository is downloaded – “Core, CSP, GFX, etc):



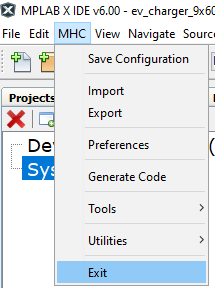
1. Enter the Project location, name and folder name as shown below:



1. Select device and choose a suitable configuration name and select Finish:



NOTE: At this point if MHC is Automatically launched, please note that you chose the wrong type of project MHC). If so, please delete the project you just created and start over.



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.

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

Graphical user interface, application

Description automatically generated

1. For “Connected Hardware Tool”, choose the J32 debugger. For Packs, choose the latest version (1.7.85) and for XC32 compiler, choose the latest version (4.10).

Graphical user interface, text, application

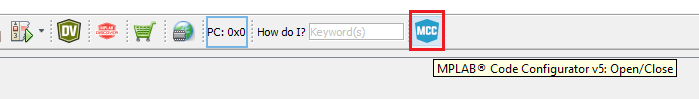
Description automatically generated

Select Apply. <Add instructions to select at91bootstrap>

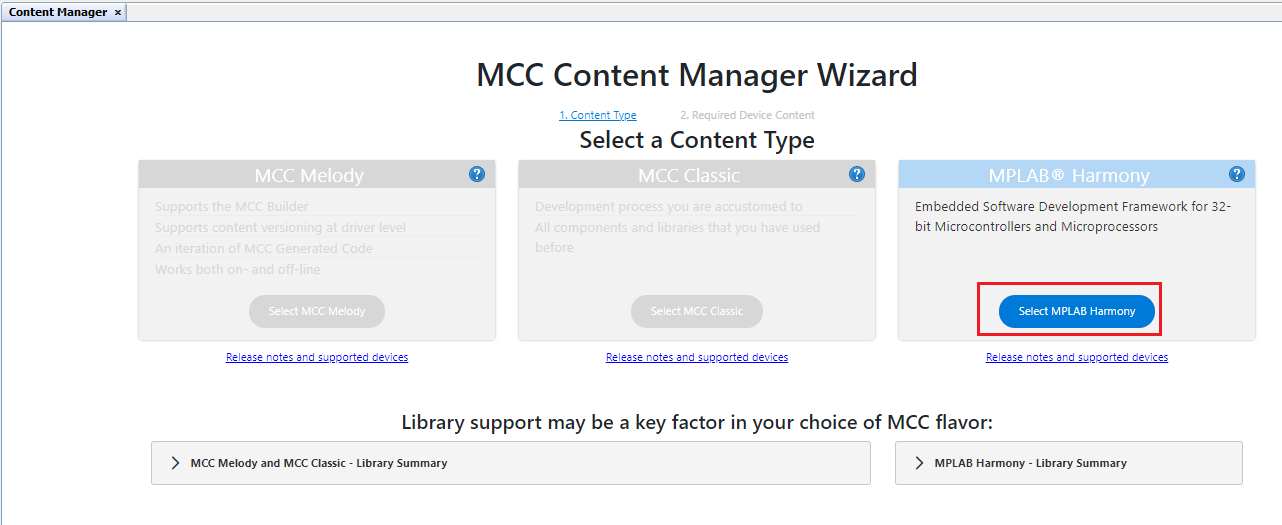
<also change jlink communications to “4-wire JTAG”>

Configuring Project Graph using MCC

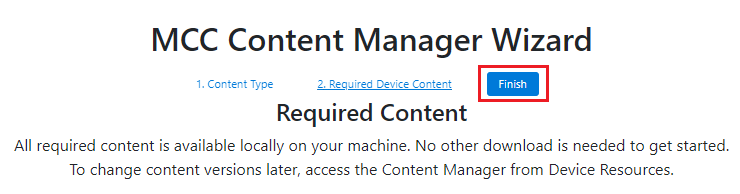
1. Launch MCC by clicking on the MCC icon on the menu bar:



Wait for the tool to launch and select MPLAB Harmony:



Click on Finish:

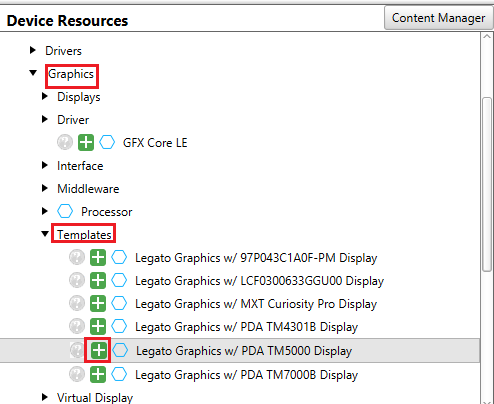


1. From the Device resources Window Select the following drivers:
2. Harmony -> Board Support Packages (BSPs) -> Click on the green + sign next to “SAM 9X60 Evaluation Kit BSP”.

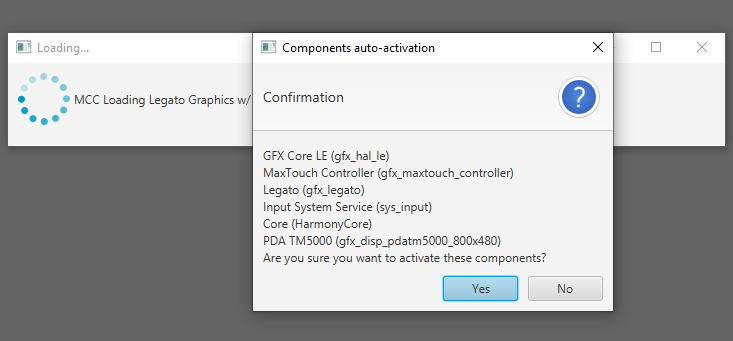
Graphical user interface, text, application

Description automatically generated

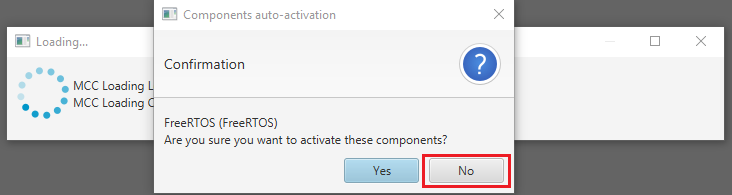
1. Harmony -> Graphics -> Templates -> Click on the green + sign next to “Legato Graphics w/ PDA TM5000 Display”



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

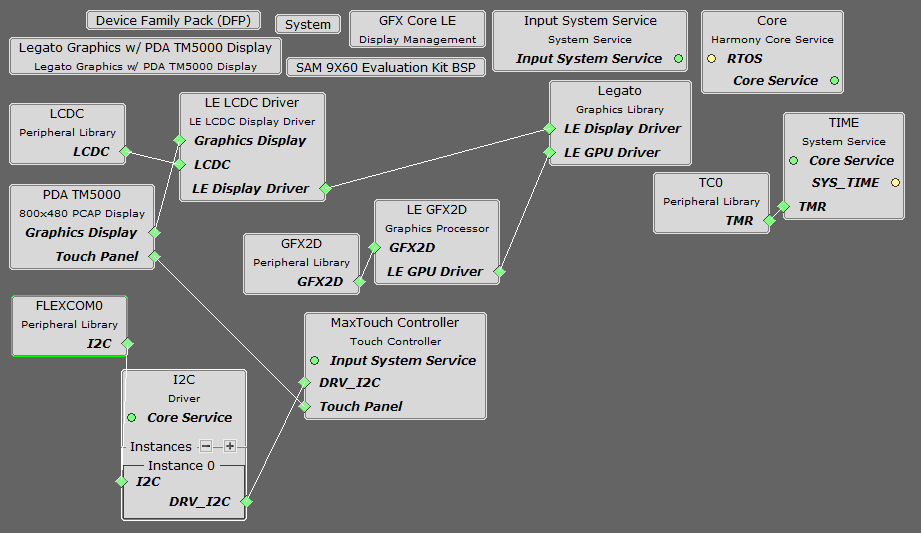


Please select “No” for FreeRTOS:

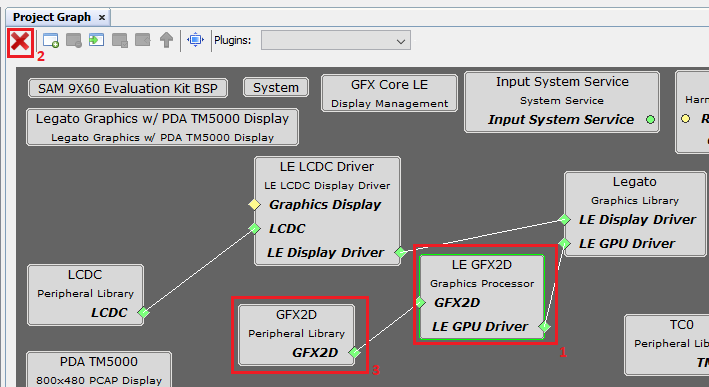


In subsequent confirmation boxes, please select “Yes”.

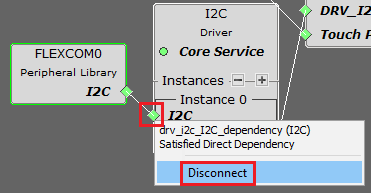
You should see a project graph generated as below



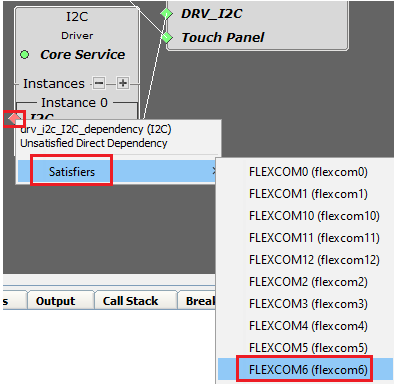
1. At this point, we will remove the 2D GPU from the project graph. Select “LE GFX2D” component and select the red cross on the top left of the Project Graph window to remove the component. Similarly remove the “GFX2D” component.



1. From the I2C driver component, right click on the green diamond and select “Disconnect”.



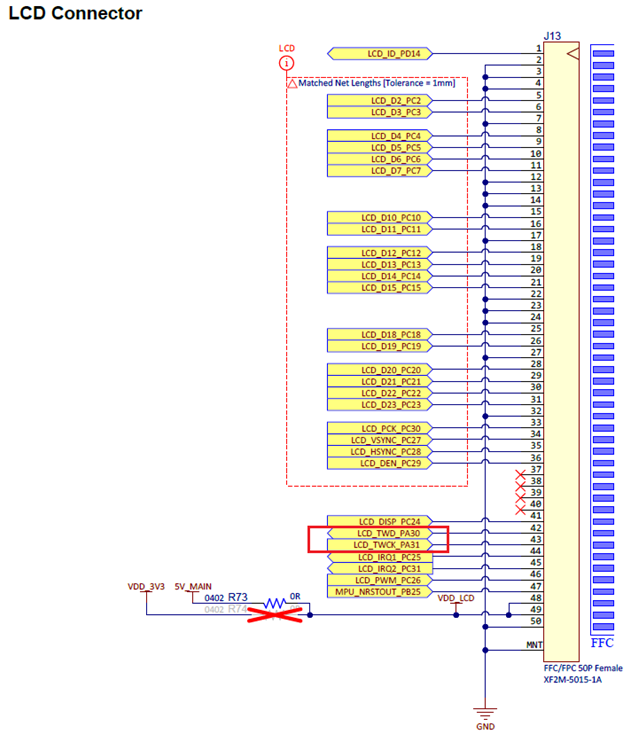
Again, right click on the green diamond for the I2C Instance, select Flexcom6 from the Satisfiers options:



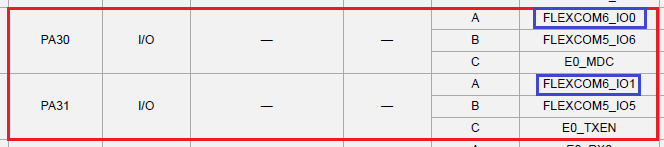
QUESTION: Why is FLEXCOM6 chosen as the satisfier? (Please try to answer this question yourself before you look at the answer in the next page)

Answer:

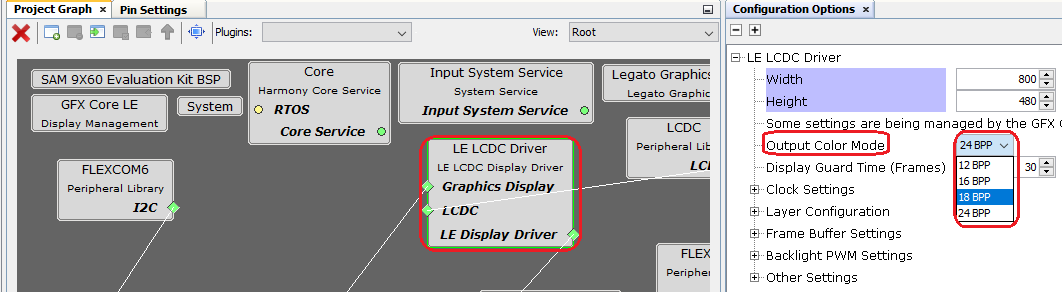
Below is schematic for the LCD Connector for SAM9X60 Curiosity board. Notice that the TWI pins used are PA30/PA31.



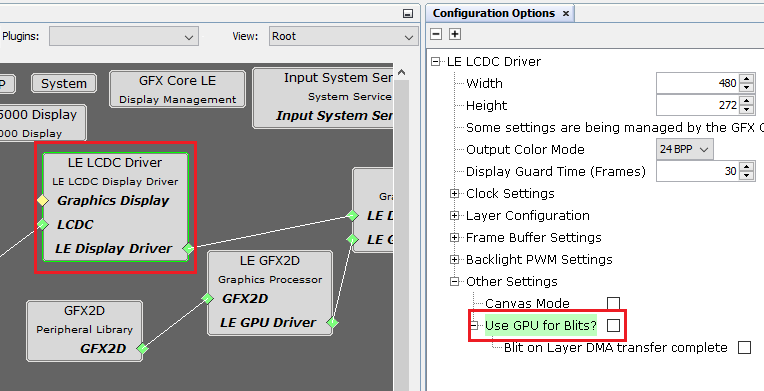
From the datasheet we can see that PA30/PA31 can be used in TWI mode (IOSet 0) using FLEXCOM6.



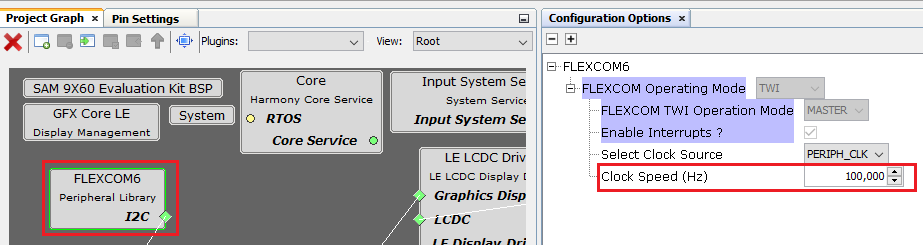
1. From the LCD pin schematics shown above, we can see that only 18 pins are used for LCD data. Therefore select “LE LCDC Driver” component from the Project graph and in the “Configuration Options”, for “Output Color Mode”, choose 18BPP.



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

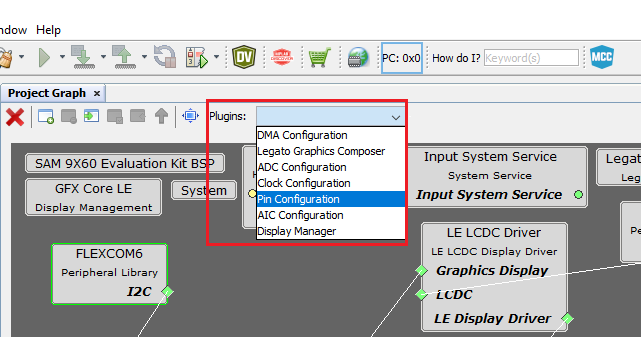


1. The I2C pins (PA30/PA31) are missing external pull up resistors. Therefore. the default TWI speed of 400kHz will not function properly. Hence let us reduce this to 100 kHz. Select FLEXCOM6 component and in the Configuration Options, for Clock Speed, please set 100 kHz.



Pin Configuration

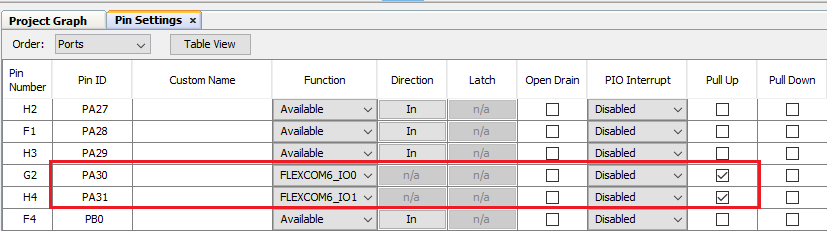
1. From the Plugins drop down menu, choose “Pin Configuration”.



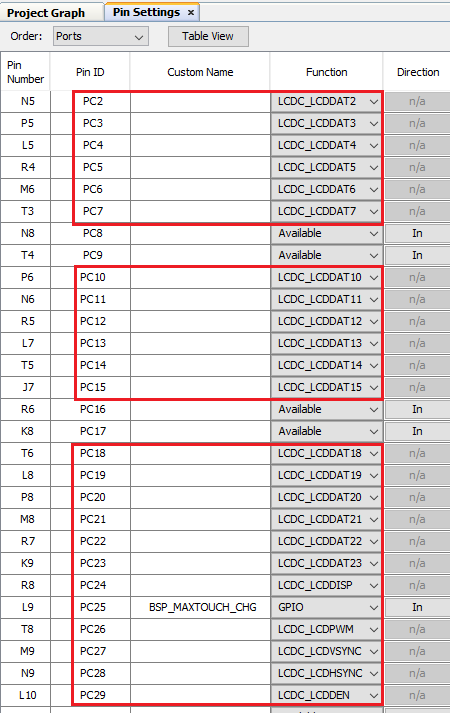
In the Pin Settings tab, from the “Order” drop down menu, choose “Ports”.



For PA30/PA31, choose FLEXCOM6\_IO0/FLEXCOM6\_IO1 for Function and select the “Pull Up” Checkbox (HINT: We select this option because external PU Resistors are missing):

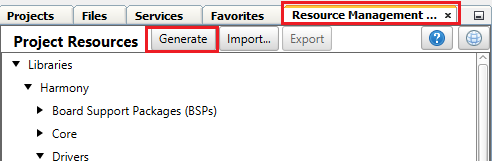


Next, we configure the LCD pins. (Refer to the LCD pins schematics shown in the page above). Set the pins to the configuration as shown below:

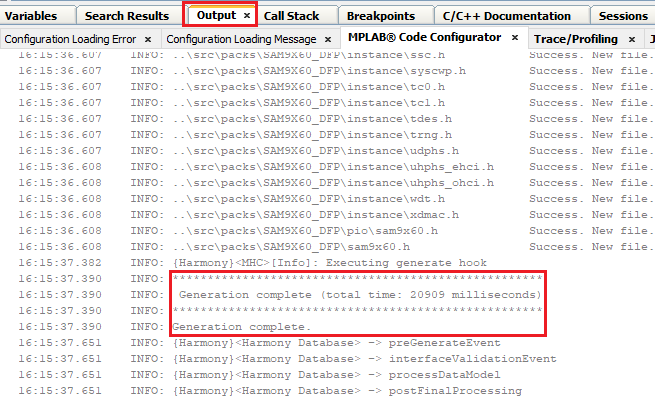


Generating Project

1. With this, all the configuration is complete. Click on the “Generate” button from Project Resources (in the Resource Management tab):



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:

Graphical user interface, text, application

Description automatically generated

Modifications to the Linker file

1. Open the ddram.ld linker file in the firmware\src\config\lcdc\_rgba8888\_mxt\_9x60\_wvga folder. Change the ram length from 0x0F000000 to 0x06f00000:

/\* Memory Spaces Definitions \*/

MEMORY

{

sram (WX) : ORIGIN = 0x300000, LENGTH = 64K /\* sram \*/

ram (LWX!R) : ORIGIN = 0x21000000, LENGTH = **0x06f00000** /\* ram \*/

ram\_nocache (!RWX) : ORIGIN = 0x20000000, LENGTH = 0x01000000 /\* ram (non-cached) \*/

rom (LRX) : ORIGIN = 0, LENGTH = 0

}

Also change the “region\_nocache” definition to as follows:

.region\_nocache (NOLOAD) :

{

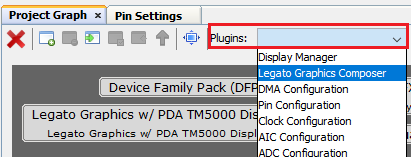
. = ALIGN(4);

\*(.region\_nocache)

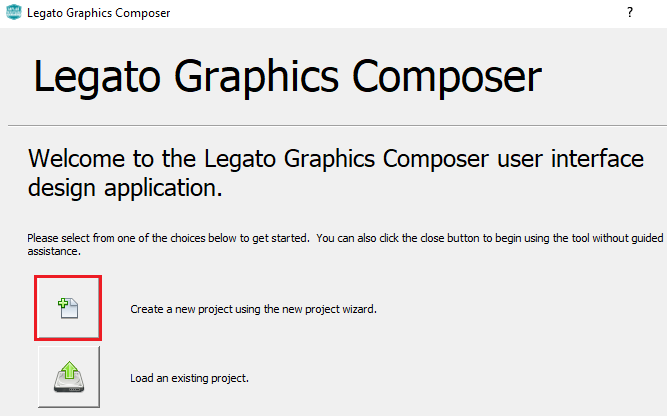
} >ram\_nocache

Legato Graphics Composer

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

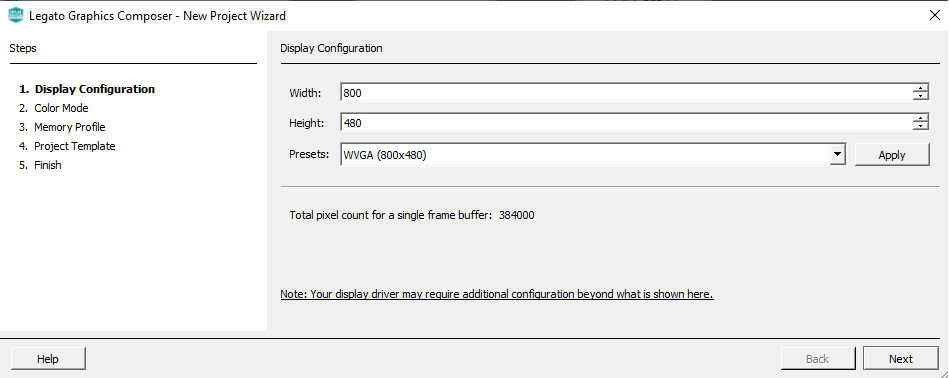


This will launch Legato Graphics Composer. Select the “Create a new project using the new project wizard icon”.

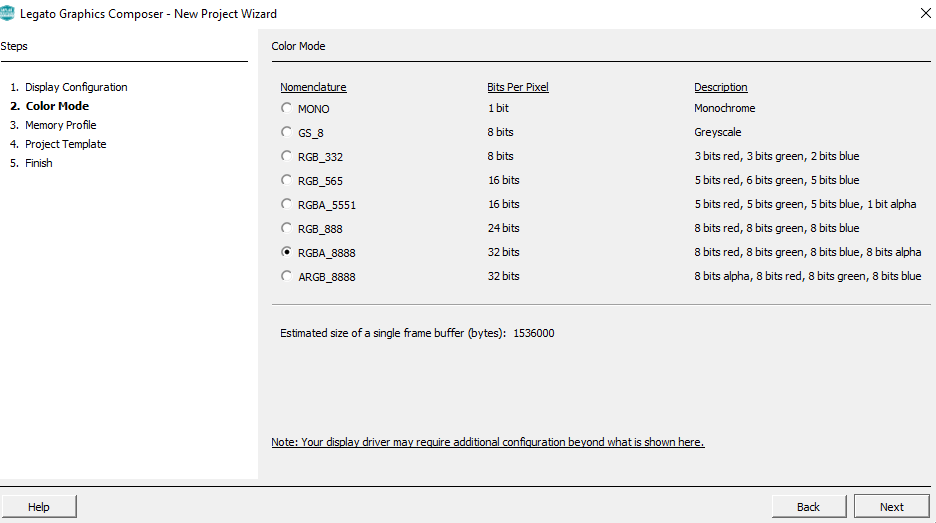


Now we will create a project template using Legato Graphics Compose wizard:

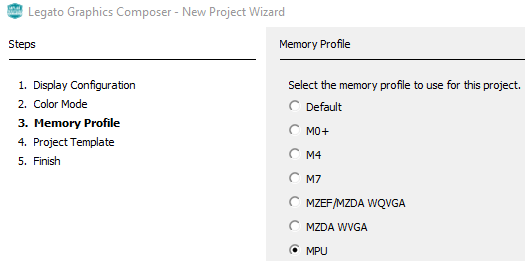
1. Enter Width, Height and Presets information as shown below and click on Next:



1. Select “RGBA\_888” for color mode and click on Next:



1. For Memory Profile select MPU:



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

<COMPOSER IS BROKEN>

<Click on Generate Again>

Add following in legato\_renderer.c:

#ifndef LE\_NO\_CACHE\_ATTR

// CUSTOM CODE - DO NOT OVERWRITE

#define LE\_NO\_CACHE\_ATTR SECTION(".region\_nocache")

// END CUSTOM CODE

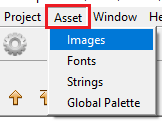
#endif

<Add instructions to rebuild and program the target. Show a pic with target displaying LegatoQS>

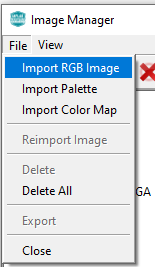
Managing Images, buttons, and Schemes

<Add instructions to <add images using Images Asset Manager>:

1. Click on “Asset” from the Composer menu amd select “Images”:

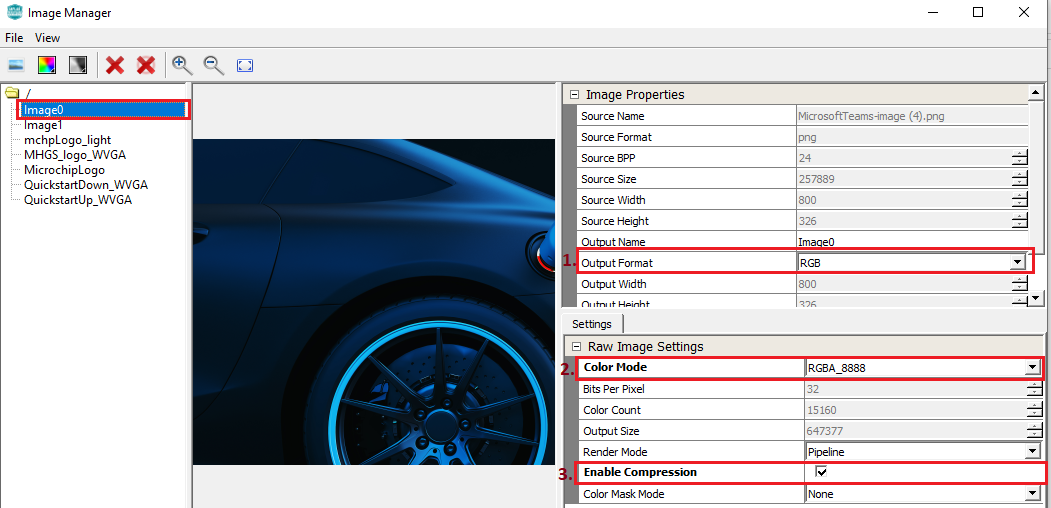


1. Click on File -> “Import RGB Image”:



Choose “Image0.png” from the “assets” folder. Similarly choose “Image1.png”, “button\_off.png”, “button\_on.png”, “Battery\_full\_horizotal.png”, “Battery\_horizontal.png”, “plus.png”, “minus.png” and “MicrochipLogo.png”

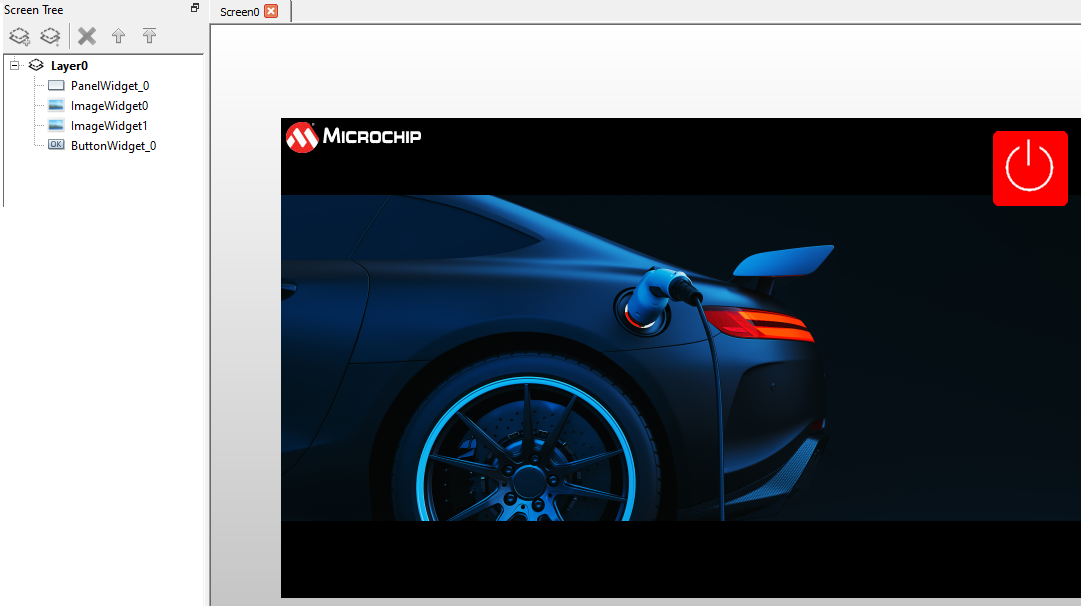
1. Next change the “Output Format”, “Color Mode” and “Enable Compression” as shown below for all 9 images:



NOTE: Please make sure you made the above changes for all 9 images – Image0.png, Image1.png, button\_off.png, button\_on.png, Battery\_full\_horizotal.png, Battery\_horizontal.png, plus.png, minus.png and MicrochipLogo.png.

< ADD instructions to add scheme, talk about transparency alpha>

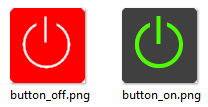
<NEXT ADD instructions to remove labels and modify rest of the widget to make screen layout like this:>



Regenerate, rebuild and debug.

**Bonus Task**

Replace the square buttons:



With round buttons:



Managing Fonts, Strings, Labels

<How to add fonts?>

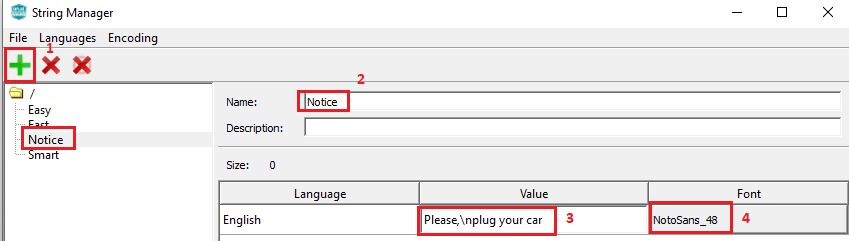
Next, let us add a string to our design:

1. Click on “Asset” option from the composer menu and click on Strings:

Graphical user interface, application, Word

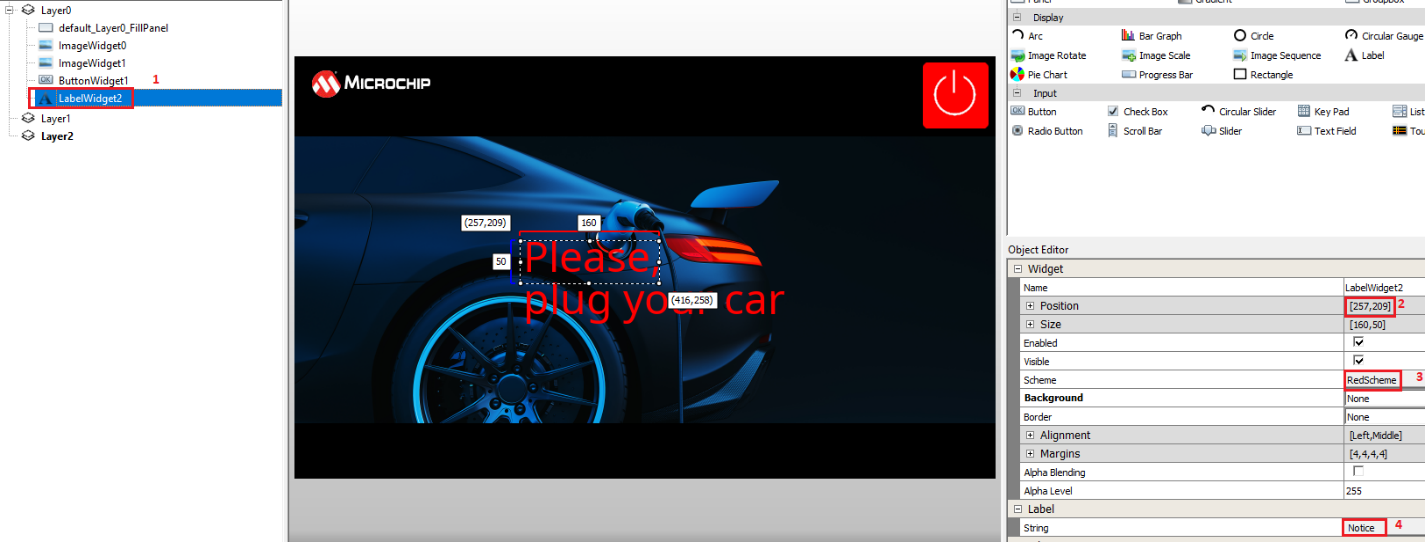
Description automatically generated

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



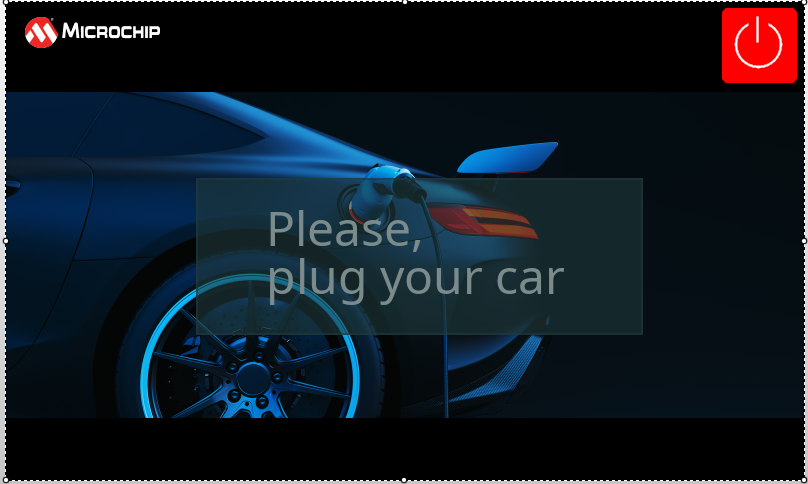
This adds a string called “Notice”, let us assign this value to the LabelWidget2.

1. Select “LabelWidget2”, set Scheme and String value as shown below:

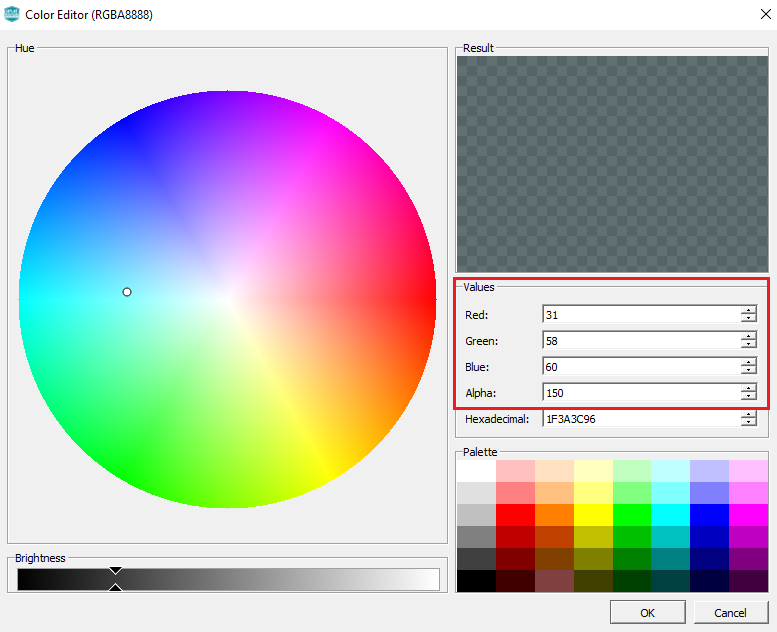


**Bonus Task**

1. Set “White” Scheme to the “LabelWidget2”.
2. Add a rectangle with transparency behind the “LabelWidget2”:



HINT: Set Base and Foreground of the Translucent scheme to the following values:



Task 2: Layers, Managing Button events and Screen Transition

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

Layers

<talk about importance of layers>

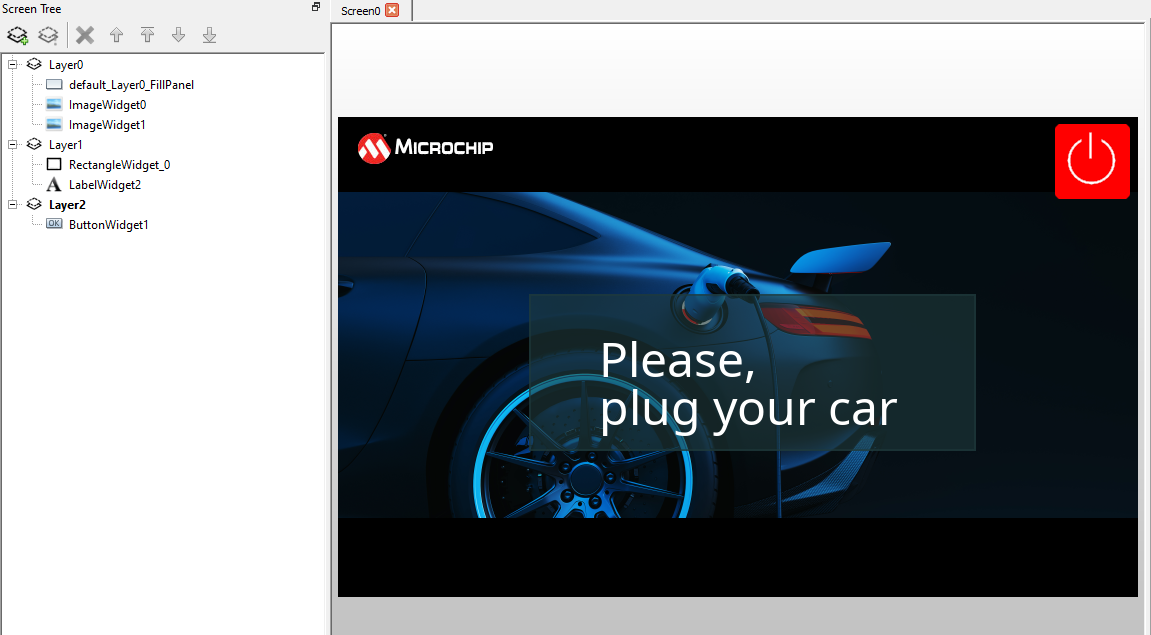
1. Move the widgets so that:

Layer0 has the 800x480 panel and the 2 image widgets

Layer1 has the rectangle with the transparency and the label widget

Layer2 has the button widget

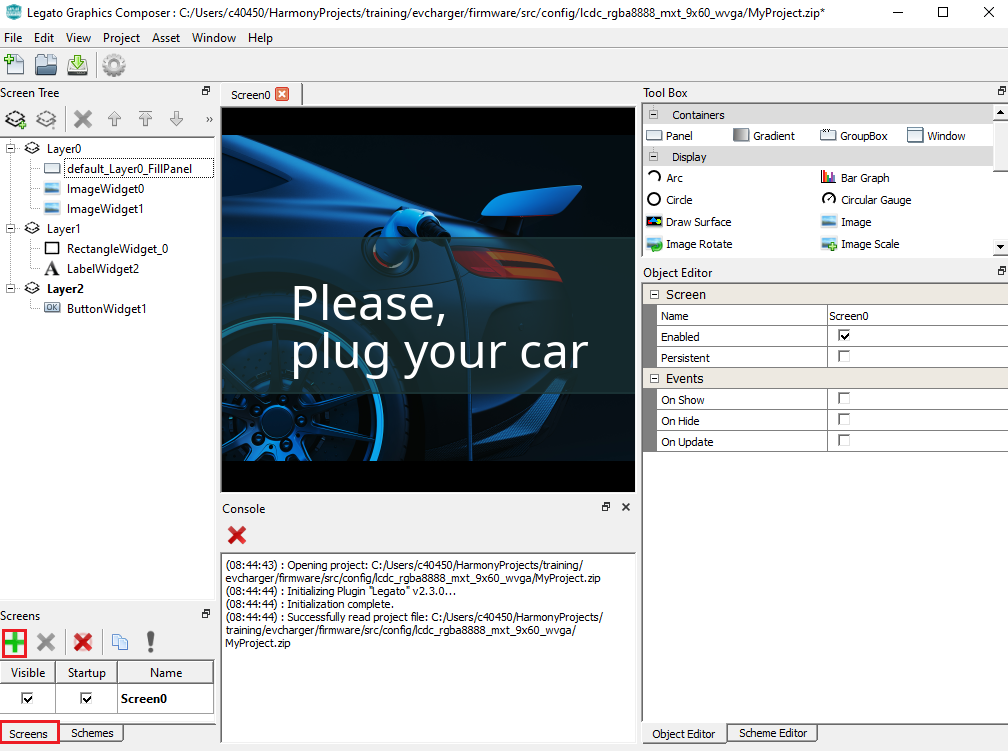
As shown in the figure below:



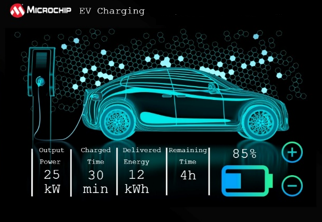
Leaving Composer open, generate MCC project again and flash the target. Check that you see the design you created on the target.

Adding New Screen

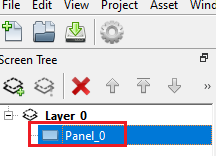
1. In legato composer, choose the “Screens” tab and click on the + sign to add new screen:



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

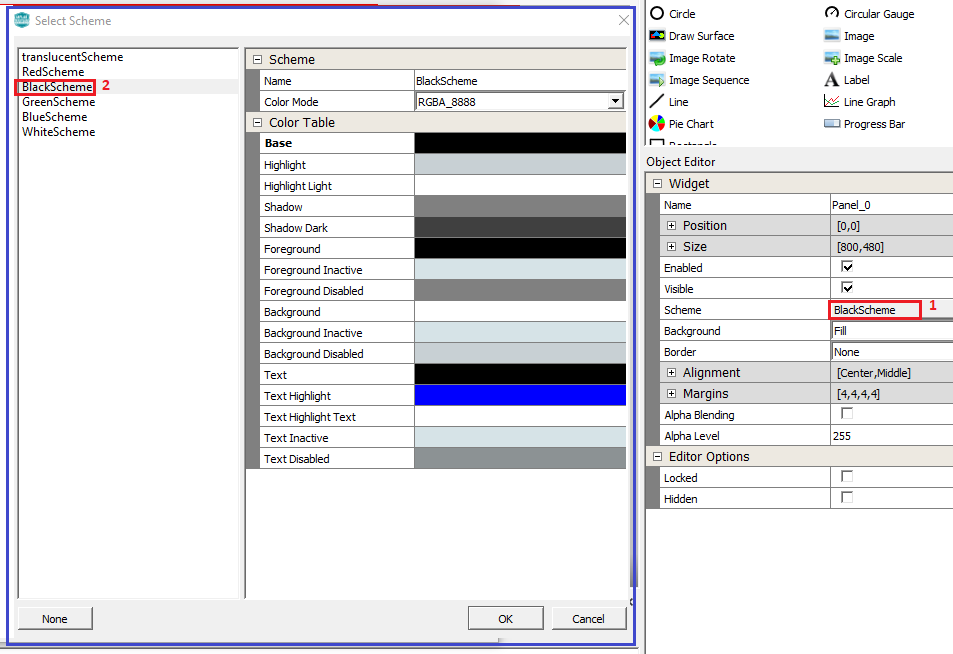
****

1. Select “Panel\_0” from the Screen Tree on the left:



1. 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 and click on OK.



Adding Images

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

Graphical user interface, application

Description automatically generated

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

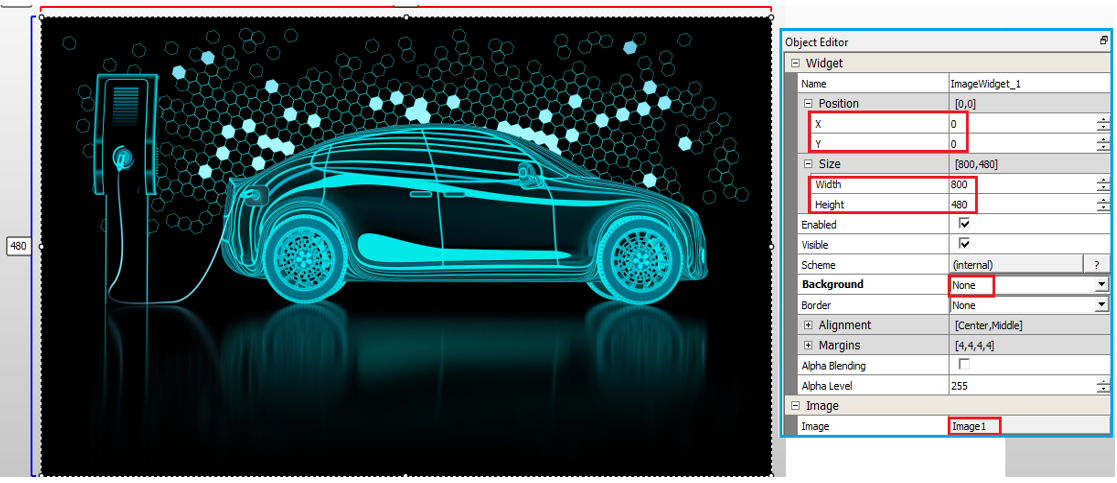
* Position – 10, 10
* Size – 144, 39
* Image – “MicrochipLogo”

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

Graphical user interface, application

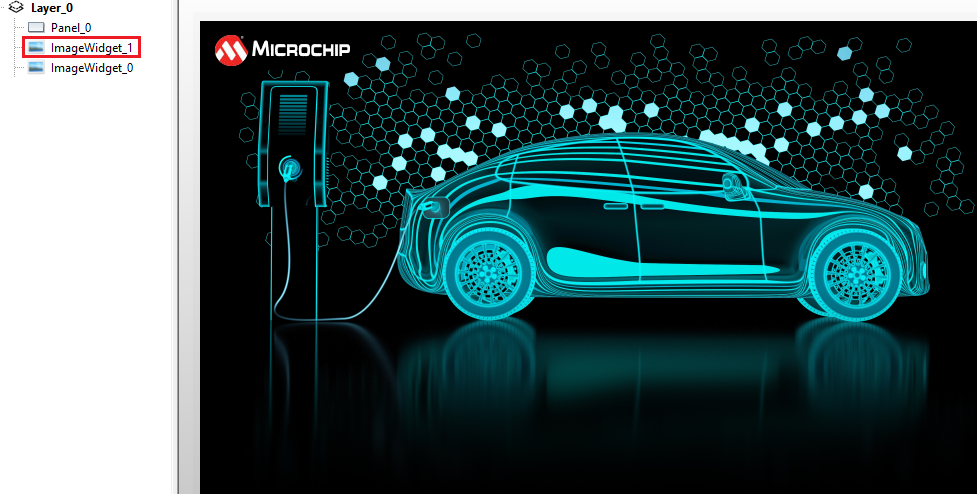
Description automatically generated

1. Drag the “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.



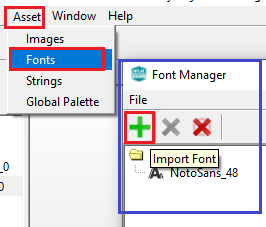


1. Drag another “Image” widget to our design and from “Object Editor”, set the values in the blue box shown below:



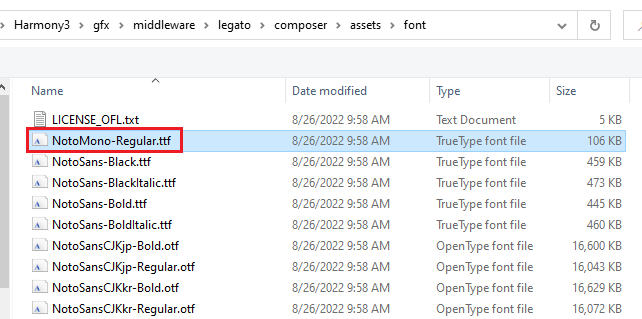
Adding New Font

1. 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 (Note: We only have font size 48 by default). To do this, click on Asset -> Fonts from the Composer main menu. From the “Font Manger” window, click the + “Import Font” icon.

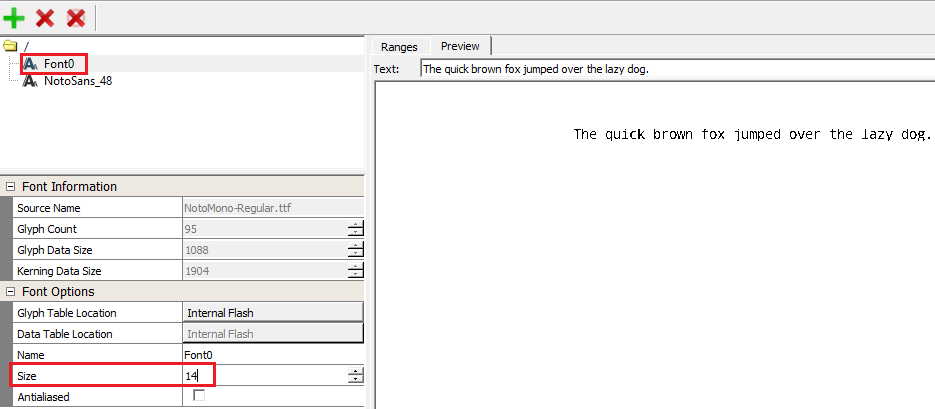


From the browse window, select NotoMono-Regular.ttf from the location – “C:\Users\c40450\Harmony3\gfx\middleware\legato\composer\assets\font”.

NOTE: Please modify the highlighted path above so that it points to where you have downloaded the Harmony framework as described in ??

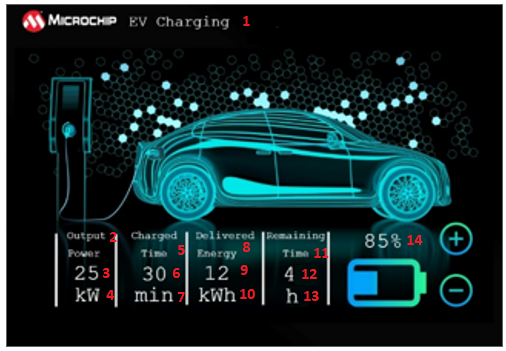


Set the font size to 14 from “Font Options” as shown below:



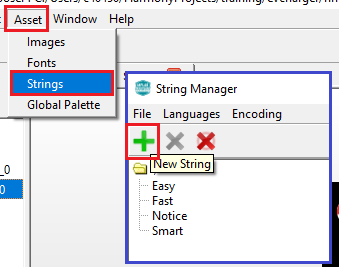
Adding Strings

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

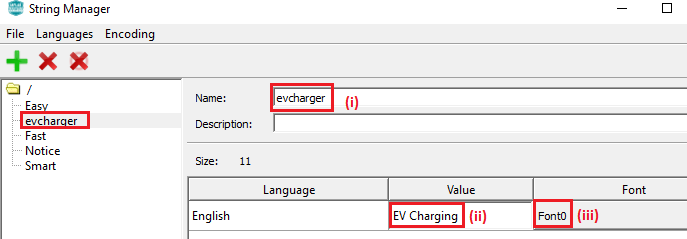


Let’s create these strings with font size 14 that we just added.

1. 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:

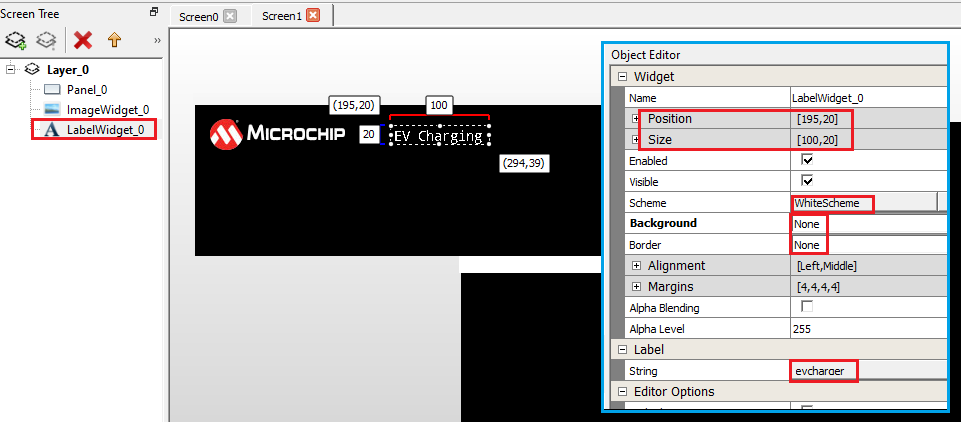


Similarly add the following strings:

|  |  |  |
| --- | --- | --- |
| **Name** | **Value** | **Font** |
| op\_label | Output\nPower | Font0 |
| ctime\_label | Charge\nTime | Font0 |
| denergy\_label | Delivered\nEnergy | Font0 |
| time\_label | Remaining\nTime | Font0 |
| pwr | 25 | Font0 |
| pwr\_unit | kW | Font0 |
| ctime | 30 | Font0 |
| ctime\_unit | min | Font0 |
| energy | 12 | Font0 |
| energy\_unit | kWh | Font0 |
| rtime | 4 | Font0 |
| rtime\_unit | h | Font0 |
| remaining\_charge | 85% | Font0 |

Adding Labels

1. 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:

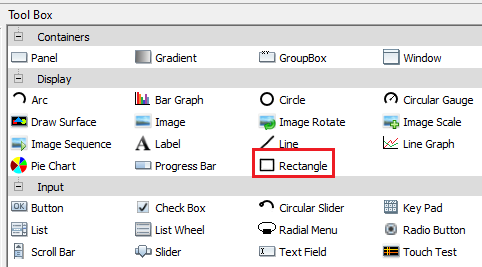


Similarly add the following Label widgets:

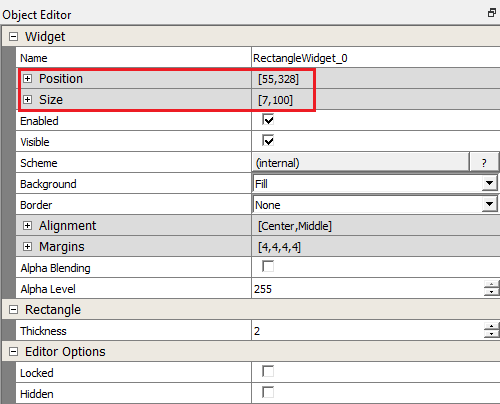
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Position** | | **Size** | | **Scheme** | **Background** | **Border** | **Label** |
| X | Y | Width | Height |
| LabelWidget\_1 | 68 | 328 | 100 | 25 | White | None | None | op\_label |
| LabelWidget\_2 | 178 | 328 | 100 | 25 | White | None | None | ctime\_label |
| LabelWidget\_3 | 275 | 328 | 100 | 25 | White | None | None | denergy\_label |
| LabelWidget\_4 | 390 | 328 | 100 | 25 | White | None | None | time\_label |
| LabelWidget\_5 | 75 | 372 | 100 | 25 | White | None | None | pwr |
| LabelWidget\_6 | 75 | 407 | 100 | 25 | White | None | None | pwr\_unit |
| LabelWidget\_7 | 181 | 372 | 100 | 25 | White | None | None | ctime |
| LabelWidget\_8 | 181 | 407 | 100 | 25 | White | None | None | ctime\_unit |
| LabelWidget\_9 | 275 | 372 | 100 | 25 | White | None | None | energy |
| LabelWidget\_10 | 275 | 407 | 100 | 25 | White | None | None | energy\_unit |
| LabelWidget\_11 | 395 | 372 | 100 | 25 | White | None | None | rtime |
| LabelWidget\_12 | 395 | 407 | 100 | 25 | White | None | None | rtime\_unit |
| LabelWidget\_13 | 580 | 324 | 100 | 25 | White | None | None | remaining\_charge |

Finishing the design

1. 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:

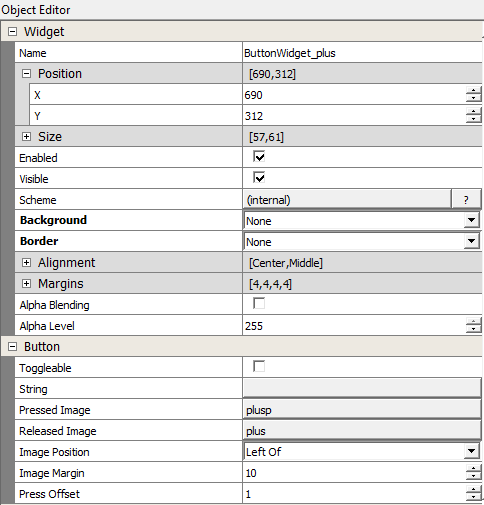
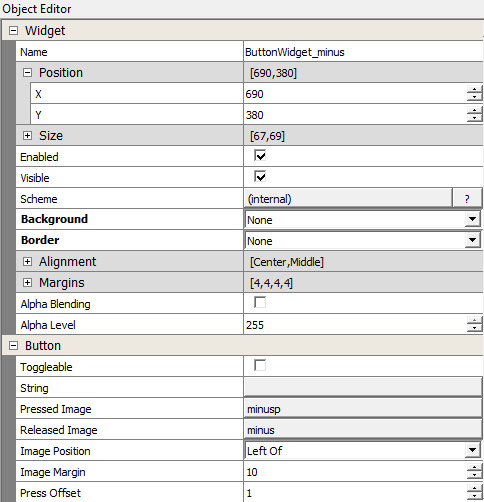
|  |  |  |
| --- | --- | --- |
| RECTANGLE | POSITION | |
| X | Y |
| RectangleWidget\_1 | 163 | 328 |
| RectangleWidget\_2 | 260 | 328 |
| RectangleWidget\_3 | 378 | 328 |
| RectangleWidget\_4 | 492 | 328 |

We now have 5 rectangles with positions and sizes as shown in the figure below:

Graphical user interface

Description automatically generated

1. Next, 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. We need to display this screen when user presses the power button on screen 0:



To do this we have to implement Event handling and screen transition. Let us learn how to do this in the next section.

**Event Management**

<Theory/Explanation>

Button Event Callback

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

Graphical user interface, text, application, email

Description automatically generated

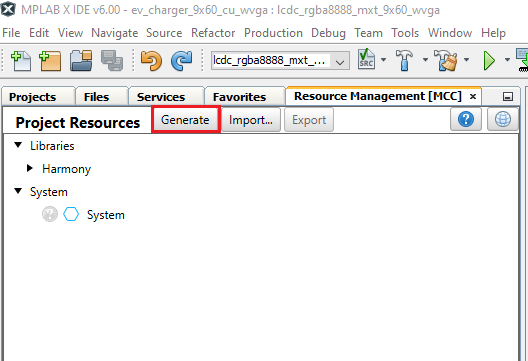
NOTE: You can also select the button widget in the layer tree and from the object editor, select “Released” for Events:

Graphical user interface, application

Description automatically generated

1. With Legato Composer open, go back to MCC tool on MPLABX IDE on “Generate” button on “Project Resources” pane.

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



This adds the following code to your project:

1. In le\_gen\_screen\_Scree0.h:

// event handlers

// !!THESE MUST BE IMPLEMENTED IN THE APPLICATION CODE!!

void event\_Screen0\_ButtonWidget1\_OnReleased(leButtonWidget\* btn);

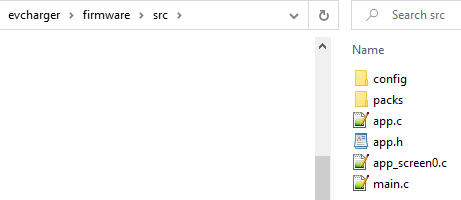
1. In le\_gen\_screen\_Screen0.c:

Screen0\_ButtonWidget1->fn->setReleasedEventCallback(Screen0\_ButtonWidget1, event\_Screen0\_ButtonWidget1\_OnReleased);

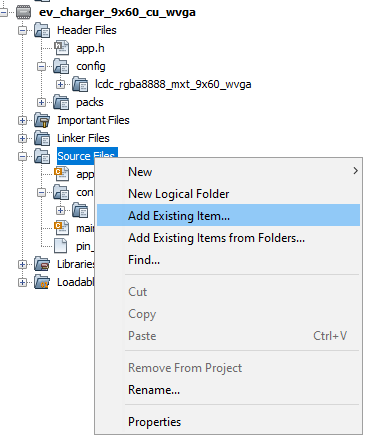
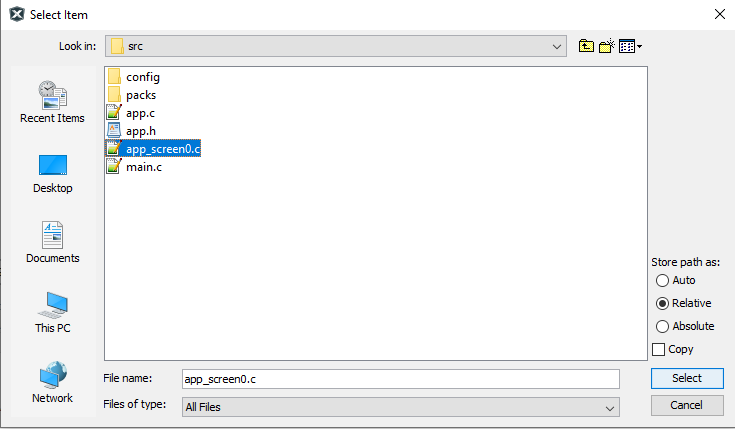
We need to implement the “event\_Screen0\_ButtonWidget1\_OnReleased(leButtonWidget\* btn)” function.

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 also to keep your code organized.

1. In “<path>\evcharger\firmware\src” folder create “app\_Scrren0.c”.



Right click on “Source Files” and select “Add Existing Item…” and choose app\_screen0.c.

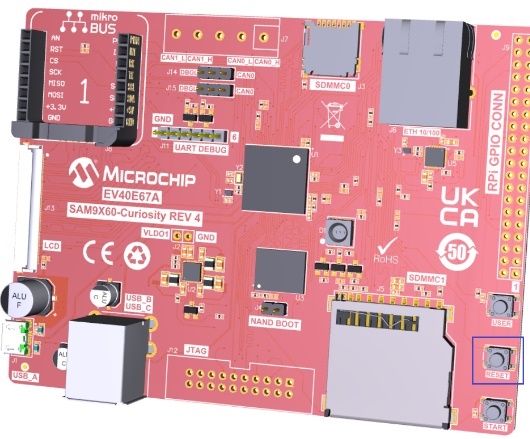
 

In app\_Scree0.c, we will implement the event\_Screen0\_ButtonWidget1\_OnReleased() function. Add the following code to app\_Screen0.c:

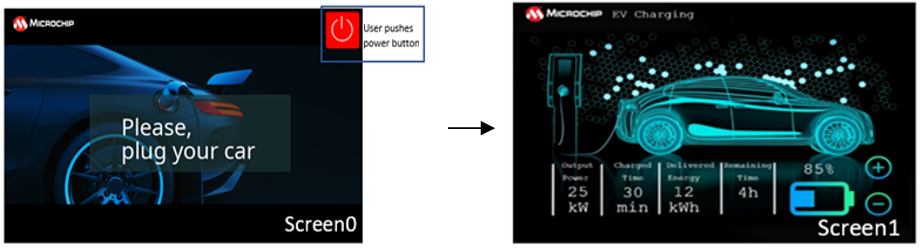
|  |
| --- |
| #include "definitions.h"  /\* Event handler for ButtonWidget1: Launch Screen1 \*/  void event\_Screen0\_ButtonWidget1\_OnReleased(leButtonWidget\* btn)  {  legato\_showScreen(screenID\_Screen1);  } |

<Description of legato\_showScreen(screenID\_Screen1); >

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



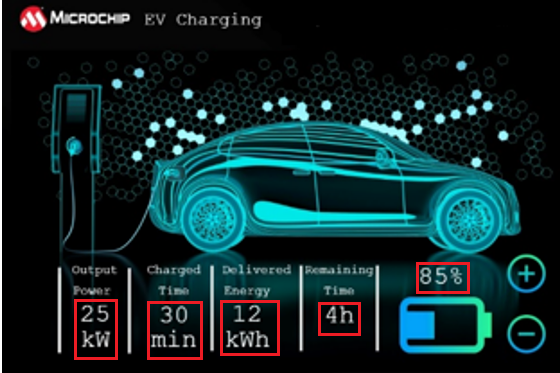
You should see Screen0 displayed on the screen. On pressing the power button on the top right of the screen, you should see Screen1.



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

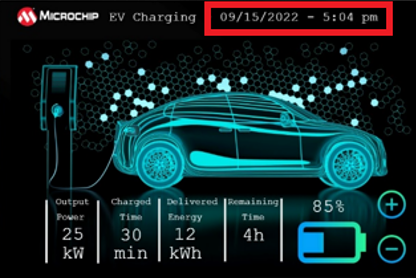
**Bonus Task**

We have used font size 14 for all the labels which makes the information displayed very small. Can you add a font size of 28 and change the labels highlighted below to this bigger font?

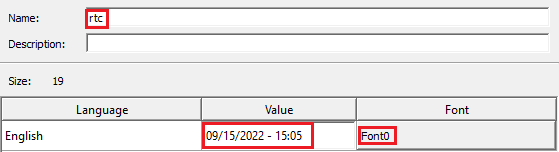


Task 3: Dynamic String and RTC

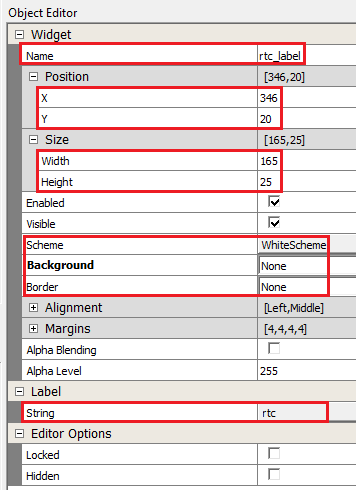
In this task, we will display the date and time using dynamic strings:



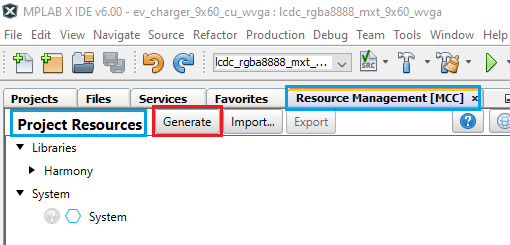
1. Let us first add a string for date / time. From Composer main menu, click Asset -> Strings. Click on **+** icon to add a new string as shown below:



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

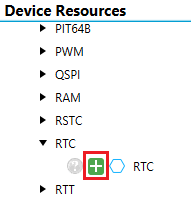


1. Then from MCC Resources Management pane, click “Generate” button:



Adding RTC support using MCC

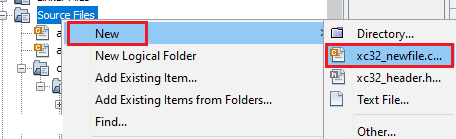
1. From “Device Resources” pane, expand Libraries -> Harmony -> Peripherals -> RTC. Click on green button as shown below which adds RTC to the project graph:



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

Writing Application Code

1. Right click on “Source Files” and click on New -> x32\_newfile.c.



Provide file name as “app\_Screen1” as shown below:



1. Delete the populated code in app\_Screen1.c and include the relevant header file 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\_rgba8888\_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]; |

Next in app\_screen1.c, add code 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\*)& Font0);  }  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);  }  } |

Declare the init\_RTC\_Label\_Screen1 and UpdateTime\_Label functions in app.h header file.

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;  } |

In APP\_Tasks() function, add the code to update the time string if application is in Screen1 context in APP\_STATE\_SERVICE\_TASKS case:

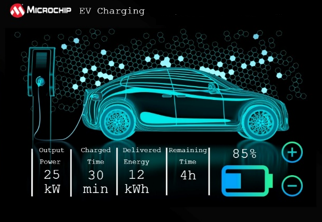
|  |
| --- |
| case APP\_STATE\_SERVICE\_TASKS:  {  if(legato\_getCurrentScreen()==screenID\_Screen1)  UpdateTime\_Label();    break;  } |

Task 4a: Sprite Animation using Image Sequence Widget and Canvas

In this task we will learn how to create sprite effect animations using Image Sequence Widget and Canvas.

**Charge Cable Animation using Image Sequence Widget**

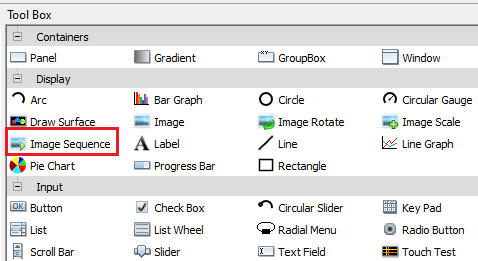
In Screen2 we will add sprite effect animation to the charge cable in the background image:

****

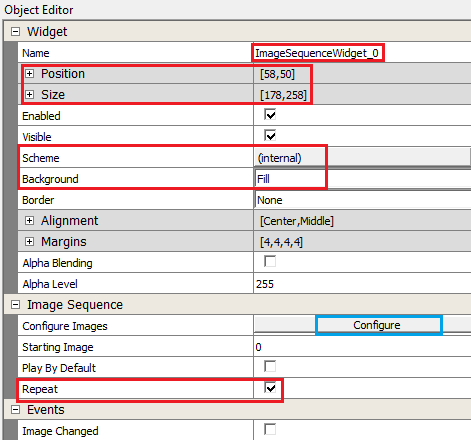
Adding Image Sequence Widget to create Sprite Effect

Image sequence widget allows user to add a number of images to create a sprite effect. The widget cycles through these images with a set delay and will repeat if configured to do so. So let us first add the images to create sprite effect and then add the image sequence widget:

1. In Graphics Composer, using “Image Manager”, add all the images in sprite\_assets folder. Remember to change Output Format to “RGB” in Image Properties and Color Mode to “RGBA\_8888” in Raw Image Settings. Also check the “Enable Compression” checkbox. NOTE: This was done in Task 1 while adding images used thus far. Please refer back to Task 1 if you need instructions for this.
2. Add the image sequence widget from the tool box to the design:

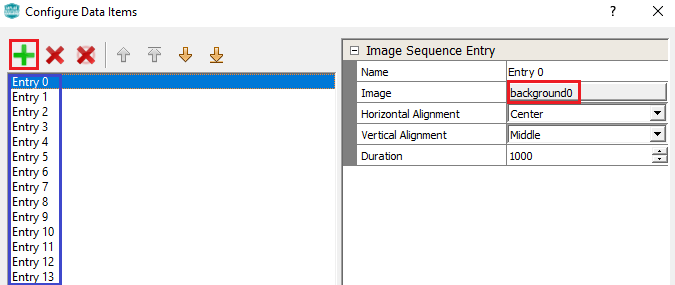


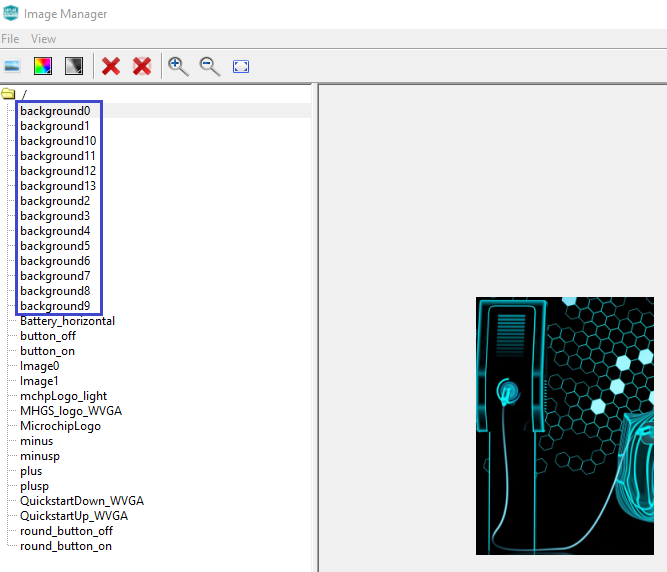
From the object editor, set the following properties to this widget:



To add images to the widget, click the “Configure” button highlighted by the blue box in the above figure.

Add the 13 images by clicking the **+** icon and setting image name in the Image Sequency Entry pane:





1. Regenerate code using MCC tool:

Graphical user interface, application, Word

Description automatically generated

1. Now add the following application code in app\_screen1.c:

|  |
| --- |
| void Update\_Screen1(void)  {  UpdateTime\_Label();  Screen1\_ImageSequenceWidget\_0->fn->showNextImage(Screen1\_ImageSequenceWidget\_0);  } |

Add a function **Update\_Screen1()** and call UpdateTime\_Label() function you wrote in the previous task. The “showNextImage” API for “Screen1\_ImageSequenceWidget\_0” cycles through all the images you added for this widget and then repeats all over again.

Declare Update\_Screen1() in app.h and call the function in APP\_Tasks() in app.c:

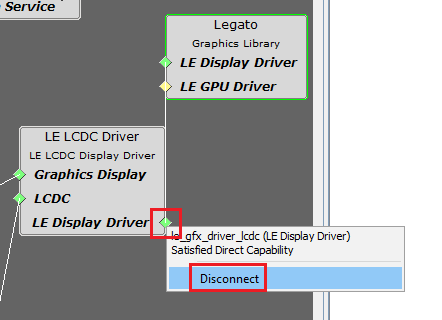
|  |
| --- |
| case APP\_STATE\_SERVICE\_TASKS:  {  if(legato\_getCurrentScreen()==screenID\_Screen1)  Update\_Screen1();    break;  } |

1. Rebuild and debug this application on the target to see the images sequence widget cycle through the 13 images that create a sprite effect for the charging cable.

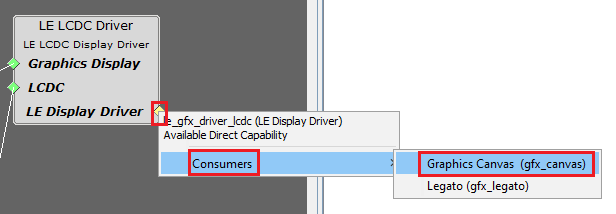
Adding Canvas Support

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

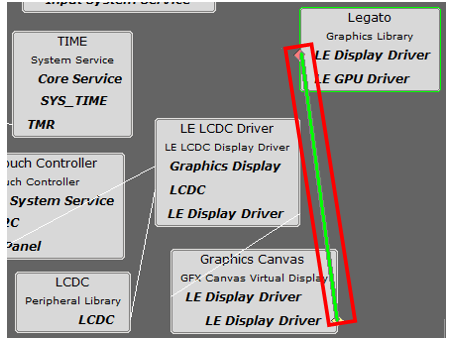
1. Right click on the green diamond on the “LE LCDC Driver” component and click on “Disconnect”. The green diamond turns yellow.



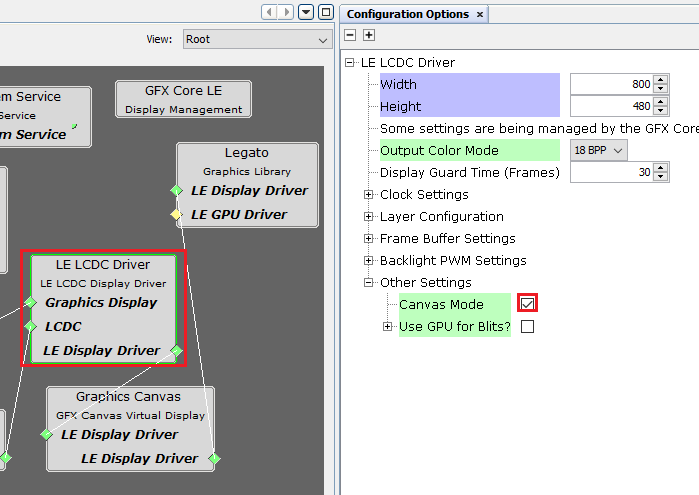
Now right-click on the yellow diamond on the “LE LCDC Driver” component and for Consumers, click on “Graphics Canvas”.



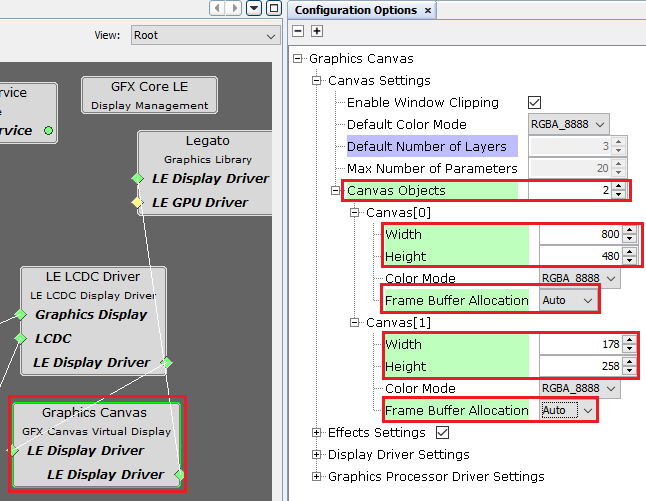
Connect “LE Display Driver” on the “Graphics Canvas” component and “Legato” component.



1. Click on “LE LCDC Driver” component and from the Configuration window, click on “Canvas Mode” checkbox.



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



1. From MCC Resources Management pane, click “Generate” button.

|  |
| --- |
| #include "peripheral/tc/plib\_tc0.h"  #include "gfx/canvas/gfx\_canvas\_api.h" |

#define BACKGROUND\_CANVAS\_ID 0

#define BACKGROUND\_LAYER\_ID 0

leImage \* imgAnim[14];

static uint8\_t anim\_cnt=0;

static uint32\_t prev\_tick, tick = 0;

/\* This function is called after period expires \*/

void TC0\_CH0\_TimerInterruptHandler(TC\_TIMER\_STATUS status, uintptr\_t context)

{

tick++;

}

void Screen1\_OnShow(void)

{

TC0\_CH0\_TimerStart();

}

void Screen1\_OnHide(void)

{

TC0\_CH0\_TimerStop();

}

void init\_anim\_images\_canvas(void)

{

imgAnim[0]= &Image1;

imgAnim[1]= &background1;

imgAnim[2]= &background2;

imgAnim[3]= &background3;

imgAnim[4]= &background4;

imgAnim[5]= &background5;

imgAnim[6]= &background6;

imgAnim[7]= &background7;

imgAnim[8]= &background8;

imgAnim[9]= &background9;

imgAnim[10]= &background10;

imgAnim[11]= &background11;

imgAnim[12]= &background12;

imgAnim[13]= &background13;

TC0\_CH0\_TimerCallbackRegister(TC0\_CH0\_TimerInterruptHandler, (uintptr\_t)NULL);

//Canvas Setup

gfxcSetLayer(BACKGROUND\_CANVAS\_ID, BACKGROUND\_LAYER\_ID);

gfxcSetWindowPosition(BACKGROUND\_CANVAS\_ID, 0, 0);

gfxcSetWindowSize(BACKGROUND\_CANVAS\_ID, 800, 480);

gfxcShowCanvas(BACKGROUND\_CANVAS\_ID);

gfxcSetPixelBuffer(BACKGROUND\_CANVAS\_ID,

800,

480,

GFX\_COLOR\_MODE\_RGBA\_8888,

(void \*) imgAnim[0]->buffer.pixels);

gfxcCanvasUpdate(BACKGROUND\_CANVAS\_ID);

}

void UpdateChargeAnime\_Canvas(void)

{

if (tick != prev\_tick)

{

prev\_tick = tick;

gfxcSetPixelBuffer(BACKGROUND\_CANVAS\_ID,

800,

180,

GFX\_COLOR\_MODE\_RGBA\_8888,

(void \*) imgAnim[++anim\_cnt]->buffer.pixels);

gfxcCanvasUpdate(BACKGROUND\_CANVAS\_ID);

gfxcCanvasUpdate(BACKGROUND\_CANVAS\_ID);

if(anim\_cnt >=14)

anim\_cnt =0;

}

}

void Update\_Screen1(void)

{

UpdateTime\_Label();

UpdateChargeAnime\_Canvas();

}

In app.c:

init\_anim\_images\_canvas();

if(legato\_getCurrentScreen()==screenID\_Screen1)

Update\_Screen1();