5054XX M-Flex Color Display TouchGFX Template User Guide

CREATED:	C. PAWLAK	DATE:	10/06/2020
CHECKED:	J. COOPER	DATE:	08/22/2024
APPROVED:	J. COOPER	DATE:	08/22/2024
ECN:	18093E	DATE:	07/31/2024

Contents

1. Software Installation

Listed below is the PC software necessary to create application software for the 5054XX M-FLEX X-Inch Color Display. Follow the directions beside each software item listed to install that piece of software.

Note: The use of asterisks (*) in file and folder names indicate varying text depending on the version of the software installed.

1. **TouchGFX Designer** is used to design the GUI (graphical user interface) for the screens displayed on the M-Flex display. Double-click on TouchGFX-*.*.*.msi to begin installing the software. Follow the directions on the screen to complete installation.

2. The **Marlin TouchGFX Template** enables TouchGFX Designer to generate applications that are compatible with the M-Flex display hardware. Copy XinchMarlinTemplate-*.*.*.tpa to C:\TouchGFX*.*.*\app\packages. The exact directory path may vary depending on where TouchGFX Designer is installed.

3. The **Marlin Programming Tool** is used in conjunction with the USB-CAN dongle to download the user's application to the M-Flex display. Open the MarlinProgToolSetup_*_*_*_Basic folder and double-click MarlinProgToolSetup_Basic.msi to begin installing the software. Follow the directions on the screen to complete installation.

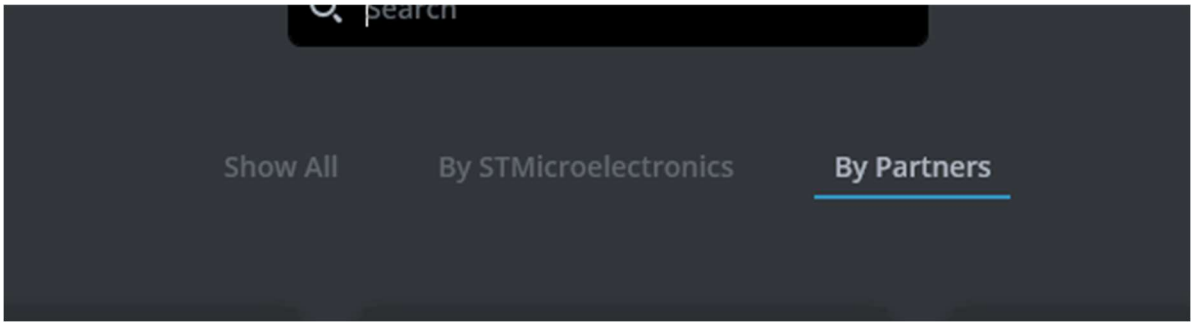
   3.1. Depending on where the Programming Tool was installed, the user guide should be located at C:\Program Files (x86)\Marlin Technologies\MarlinProgTool_Basic\UserGuide_*.pdf. Open the user guide for reference in the next step.

4. Install the driver for the appropriate **USB-CAN dongle,** as directed in the Programming Tool User guide.

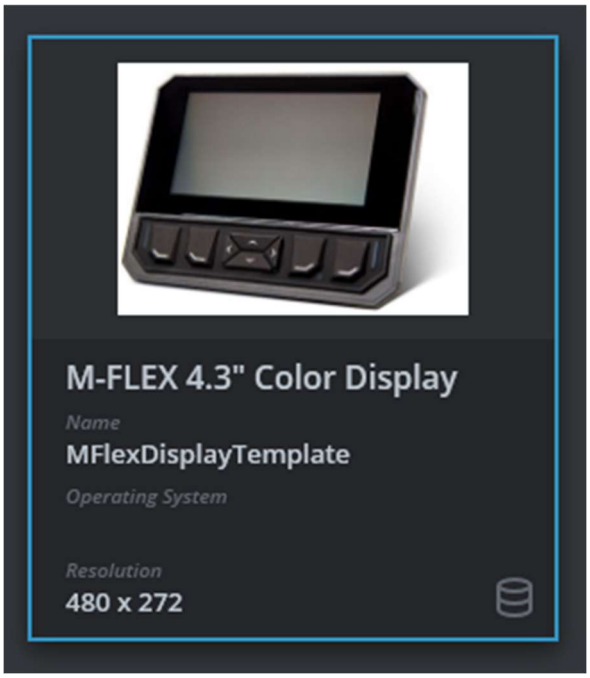
5. The **Marlin Hex Parser** (used to generate the required s19 files) requires the .NET 7.0 (or newer) desktop runtime. If this is not installed, it can be downloaded here.

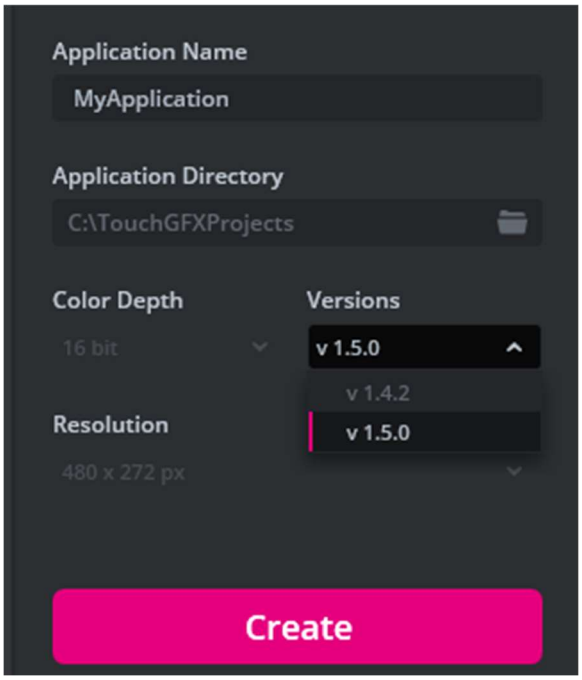
2. Creating a Project in TouchGFX Designer Using the Template

The Marlin TouchGFX template can be found under the "*By Partners*" tab when creating a new project in TouchGFX.



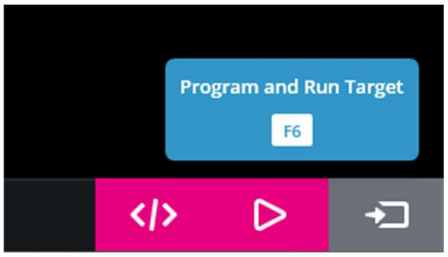
Look for the option titled "*M-FLEX Color Display*" and be sure to select the most recent version on the right-hand side.





3. Loading the Display Module with Software

The 5054xx display module requires an .S19 file to be loaded. An .S19 file can be created by clicking the "Program and Run Target" button in the bottom right-hand corner of the window.



If you see the below lines in the build window, then the .s19 creation was successful.

```
Support/Parser/MarlinHexParser.exe -HWID 0x0299 -start 0x08008000 -end 0x91FFFFFF -checksumaddress 0x080082C0 -g -b build/target.hex
-------------------------------------
|     File Conversion Complete     |
-------------------------------------
Support/Parser/MarlinHexParser.exe -HWID 0x0299 -start 0x08008000 -end 0x08FFFFFF -checksumaddress 0x080082C0 -g -b build/intflash.hex
-------------------------------------
|     File Conversion Complete     |
-------------------------------------
```

To load the .S19 file, the Marlin Programming tool must be used. For more information on the Marlin programming tool, refer to the user guide located at:

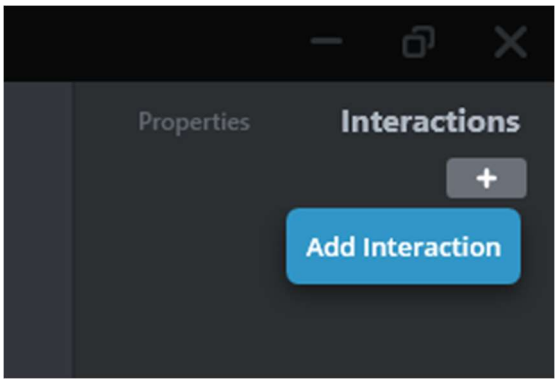
C:\Program Files (x86)\Marlin Technologies\MarlinProgTool\UserGuide.pdf

The below instructions may also serve to help.

1) Open the Marlin programming tool
2) Click the *Browse* button and navigate to the "*${Project_Location}/build/*" folder.
3) Here there are two options to choose from:
   a) Choose "*target.s19"* for the full program. (*This option will take the longest*)
   b) Choose "*intflash.s19"* for only the program space. (***This is meant for debug only.*** *This file will load much faster than* "target.*s19"*. This will not load image assets and may result in unpredictable behavior if images have changed since the last build*)
4) Click the "*Search*" button and ensure that the display shows up and is selected in the drop down menu.
5) Click the "*Program"* button to download the application to the display module.
   a) Loading time is largely dependent on the size and number of image assets used in the application.

4. Hardware Buttons

The display module's buttons can be used via the interactions tab in the top right-hand corner of the TouchGFX editor.



When using the display modules buttons, select "*Hardware button is clicked*" as the interaction trigger, and refer to figure X for what to select under the "*Choose button key"* option.

Tip: When choosing the "*Action*", if the desired behavior is, for example, an output to turn on, choose "*Call new Virtual Function*" as the action. This will enable the ability to call underlying hardware functions when a button is pressed. The generated virtual function can be found in the view base cpp file that gets generated for that screen.

5. IO and LED Driver Functions

All IO and LED driver functions can be accessed by including the "*bsp.h*" header file.

5.1 LCD Screen

The LCD back light brightness can be modified by calling:

```
void setKeypadBacklightIntensity(uint32_t);
```

The `uint32_t` parameter is a number from 0-1000 that represents a duty cycle percentile (0.1% per bit).

5.2 Keypad LEDs

There are 2 keypad LEDs on either side of the keypad, an RGB and solid Red LED. The individual sides cannot be independently controlled. The functions for controlling these LEDs are:

```
void setColorLedRGB(uint32_t, uint32_t, uint32_t);
void setColorLedRed(uint32_t);
void setColorLedGreen(uint32_t);
void setColorLedBlue(uint32_t);
void setRedLedIntensity(uint32_t);
```

The `uint32_t` parameters are numbers from 0-1000 that represent a duty cycle percentile (0.1% per bit).
For the **`setColorLedRGB`** function, the parameters are in the order of Red, Green, Blue.

5.3 Outputs

There are 2 PWM half-bridge outputs on the display. They can be controlled using the following functions:

```
void setOutput_1(uint32_t);
void setOutput_2(uint32_t);
```

Again, the `uint32_t` parameters are numbers from 0-1000 that represent a duty cycle percentile (0.1% per bit).

5.4 Inputs

There are 4 inputs on the display module. Inputs 1 & 2 are both digital and analog. Inputs 3 & 4 are only digital. All inputs come with a toggleable internal pull up to V_{in}. The following functions can be used to interact with the inputs:

```
void input1_enablePullUp(void);
void input1_disablePullUp(void);
void input1_enablePullDown(void);
void input1_disablePullDown(void);
void input1_set37vRange(void);
void input1_set10vRange(void);
int input1_ReadDigitalStatus(void);
int input1_ReadAnalogStatus(void);

void input2_enablePullUp(void);
void input2_disablePullUp(void);
void input2_enablePullDown(void);
void input2_disablePullDown(void);
void input2_set37vRange(void);
void input2_set10vRange(void);
int input2_ReadDigitalStatus(void);
int input2_ReadAnalogStatus(void);

void input3_enablePullUp(void);
void input3_disablePullUp(void);
int input3_ReadDigitalStatus(void);

void input4_enablePullUp(void);
void input4_disablePullUp(void);
int input4_ReadDigitalStatus(void);
```


6. EEPROM

The display module contains a 512Kb EEPROM. Driver functions for interfacing with the EEPROM can be used by including the *"25lc512t.h"* file.

The SPI peripheral used for the EEPROM is a shared resource. So before accessing the EEPROM, *"EE_InitSpi"* must be called:

```
void EEP_InitSpi();
```

Use the following functions to read/write data to the EEPROM, or check the EEPROM status:

```
uint8_t EEP_ReadStatus();
uint8_t EEP_ReadByte(uint16_t Address);
void EEP_ReadBlock(uint16_t Address, uint8_t *pData, uint16_t Size);
void EEP_EraseChip();
void EEP_WriteByte(uint16_t Address, uint8_t Data);
uint8_t EEP_WriteBlock(uint16_t Address, uint8_t *pData, uint16_t Size);
```

where

- `Address` is the EEPROM address (starting at 0)
- `*pData` points to a block of bytes to be read or written to
- `Data` is a byte of data to write
- `Size` is the number of bytes of data to be read/written.

7. Tasks

The TouchGFX template utilizes CMSIS V2 FreeRTOS for runtime task management. Tasks are initialized in the *"freertos.c"* file. Task files are located in the *"Src/Tasks"* folder. Each task should get its own source file for the sake of organization.

Each task requires a unique *handle* and a set of *attributes*. Handles and Attributes are initialized in the *"freertos.c"* file.

The template includes a user CAN task which can be used as an example for initializing new tasks:

```
/* Definitions for userCANTask */
osThreadId_t userCANTaskHandle;
const osThreadAttr_t userCANTask_attributes = {
  .name = "userCANTask",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityNormal,
};
```

```
userCANTaskHandle = osThreadNew(StartUserCANTask, NULL, &userCANTask_attributes);
```


8. Semaphores

Semaphores are used to control access to shared resources. Semaphores are initialized in the *"freertos.c"* file. Semaphore files are located in the *"Src/Semaphores"* folder. Each semaphore should ger it's own file for the sake of organization.

Each semaphore requires a *"Handle"* and *"Attributes"*. Handles and Attributes are initialized in the *"freertos.c"* file.

The template includes a pre-existing semaphore that can be used as an example:

```
/* Definitions for adcSem */
osSemaphoreId_t adcSemHandle;
const osSemaphoreAttr_t adcSem_attributes = {
  .name = "adcSem"
};

adcSemHandle = osSemaphoreNew(1, 1, &adcSem_attributes);
```


9. CAN

CAN functions can be utilized by including the *"marlin_can_lib.h"* header file. There are only 2 functions needed to utilize CAN.

```
enum marlinJ1939_error_codes CAN_DequeueMessage(struct can_msg *msg, uint32_t ms);
enum marlinJ1939_error_codes CAN_QueueMessage(struct can_tx_msg *msg);
```

These functions both require the operating system to be active in order to work. Do not call these functions before the operating system has started. This can be ensured by only calling these functions from within Tasks.

10.1 Receiving CAN messages

"CAN_DequeueMessage" is a blocking function and will yield the task to the OS scheduler for the specified timeout or until a CAN message is received. The received message will be returned on the passed `*msg` parameter. If the timeout is reached without a new message being received, `*msg` will contain the last received message.

This function must never be called from within the Display Task (responsible for drawing the screen) **given its yielding behavior.**

Note: It is recommended to call *"CAN_DequeueMessage"* with a timeout value of *"osWaitForever"*, as this prevents the return of stale message data.

10.1 Transmitting CAN messages

"*CAN_QueueMessage*" is safe to call from any task as it does not yield.

It is not necessary for the `can_tx_msg` parameter to persist past the calling function. A deep copy of the struct is made when it is inserted into the queue.

10. Clock

The display module has a built-in Real-Time Clock (RTC) that can be used to track time and date. The RTC can be utilized by including the *"rtc.h"* header file. There are 4 functions that are used to interact with the RTC:

```
HAL_StatusTypeDef HAL_RTC_GetDate(RTC_HandleTypeDef *hrtc,
                                  RTC_DateTypeDef *sDate, uint32_t Format);
HAL_StatusTypeDef HAL_RTC_GetTime(RTC_HandleTypeDef *hrtc,
                                  RTC_TimeTypeDef *sTime, uint32_t Format);
HAL_StatusTypeDef HAL_RTC_SetTime(RTC_HandleTypeDef *hrtc,
                                  RTC_TimeTypeDef *sTime, uint32_t Format);
HAL_StatusTypeDef HAL_RTC_SetDate(RTC_HandleTypeDef *hrtc,
                                  RTC_DateTypeDef *sDate, uint32_t Format);
```

Where:

- `*hrtc` will always be `&hrtc`
- `*sDate/*sTime` is the struct that will store the Date/Time
- `Format` will be either `RTC_FORMAT_BIN` for Standard Binary format or `RTC_FORMAT_BCD` for Binary Coded Decimal format.

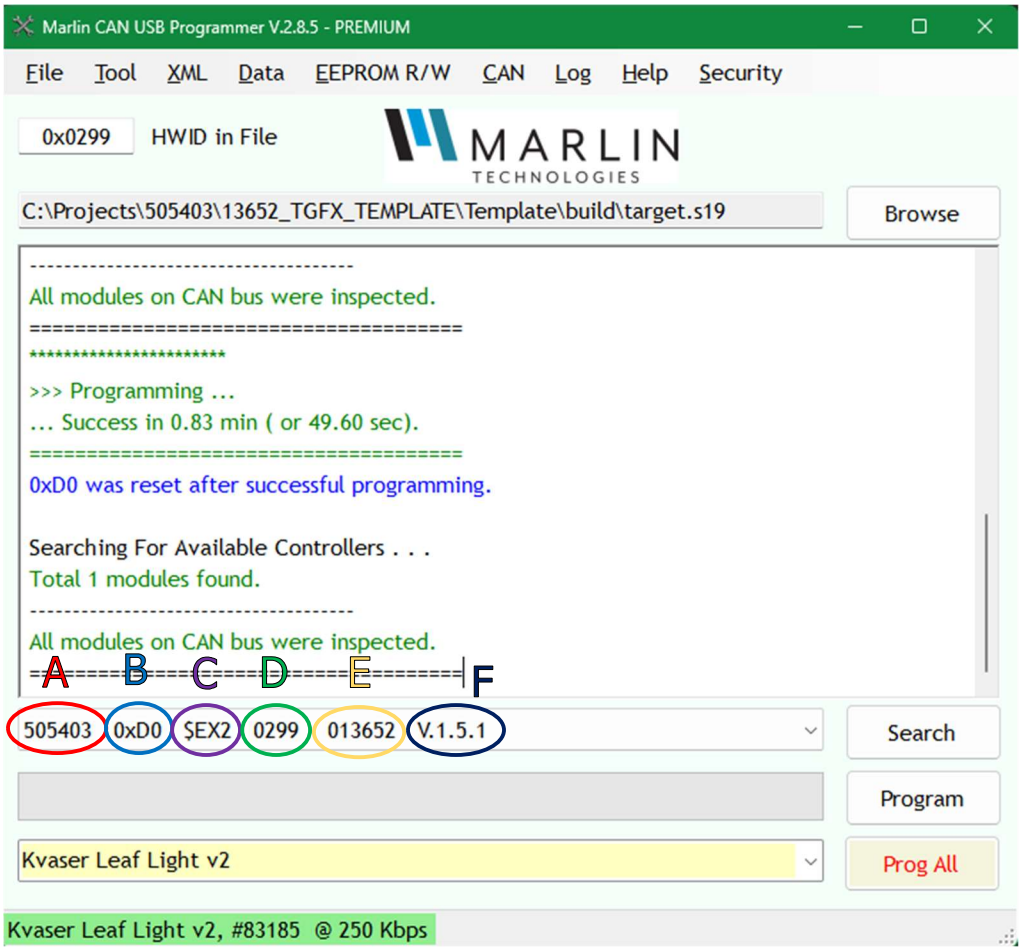
Note: When using the Get functions, they both must be used at the same time and in the order of Time then Date. Not doing so will leave the RTC in a bad state and result in bad future readings.

Example:

```
// Init current time
HAL_RTC_GetTime(&hrtc, &lastTime, RTC_FORMAT_BIN);
HAL_RTC_GetDate(&hrtc, &lastDate, RTC_FORMAT_BIN);
```


11. Changing Application Software Identifiers

Application Identifiers are located in the "*J1939_Def.h*" header file and in the "*bootloader_helper.h*" header file.



A. Module number is located in the "*bootloader_helper.h*" header file:

```
#define PROJECT_NUMBER              505403
```

B. J1939 Source address is located in the "*J1939_Def.h*" header file:

```
#define SOURCE_ADDRESS              0xD0
```

C. Software Revision is located in the "*J1939_Def.h*" header file:

```
#define SW_REV                      {'$','E','X','2',' ',' '}
```


D. Hardware ID is located in the "*bootloader_helper.h*" header file:

```
#define MODULE_HWID                    0x0299
```

Do Not Change this number. Changing this number will make reprogramming of the module difficult and/or impossible.

E. Software Number is located in the "*J1939_Def.h*" header file:

```
#define SW_NUMBER        {'0','1','3','6','5','2'}
```

F. TouchGFX Template Version is located in the "*J1939_Def.h*" header file:

```
#define SW_ID            {1, 5, 1}
```

While not critical, like Hardware ID is, it's recommended not to modify the Template version as it will make troubleshooting with Marlin representatives more difficult should such help be required.

12. Troubleshooting Tips and Common Pitfalls

12.1 Clean Building

Certain build errors and unexpected functionality may be resolved by deleting the *build* folder (in the application's top-level folder) prior to clicking Run Target in TouchGFX Designer. This forces a clean build of the application, in case TouchGFX Designer fails to rebuild a portion of the application that the user has modified since the previous build. Remember to back up any necessary files (such as .s19 files) before deleting the *build* folder.

12.2 Entering Bootmode

If there is an error in the user application that prevents the display module from communicating with the Programming Tool, the display can be forced into bootloader mode and then reprogrammed. To enter bootloader mode, first power down the display module. Then power up the module while holding down the following three buttons:

Figure 1

When the display is in bootloader mode, the screen is black, and the blue or red LEDs will be turned on for about 4 seconds. After which the red LEDs will turn off.

Note: while the red LEDs are ON, the display will not respond to CAN requests.

If "Hardware ID Mismatch!" is displayed by the Programming Tool, check that MODULE_HWID in Inc\bootloader_helper.h is defined as follows:

The hardware ID defined in the application program must match the hardware ID of the bootloader. The Programming Tool checks for matching hardware IDs to ensure that compatible software modules are programmed onto modules.

12.3 Screen Blinking/Unresponsive CAN

If after powering on the display it appears to blink at a regular period and is largely unresponsive (on both CAN and the UI), then it might be watchdog resetting. When certain faults are encountered, the display will enter an infinite loop and wait for the watchdog timer to

force a system reset. Depending on the code, this can be a perpetual reset loop, which results in the observed screen blinking. There are 2 common causes for this, hard faults and operating system related faults.

If this is happening, the only way to reprogram will be to put the display into boot mode. Refer to the above section 13.2.

12.3.1 Hard Fault - Array out of bounds

There are 2 possible causes for array out bounds faults. Direct array accesses and TouchGFX asset accesses.

Ensure any arrays or lists are not being accessed beyond their limits. Under such circumstances, the microcontroller doesn't know how to handle it. Ensure that any array accesses are within bounds. Violations here can often happen in *while* or *for* loops where a hardcoded value wasn't changed when an array size was modified.

TouchGFX generates "key" values for all used text and image assets. These keys can be found in the following files:

```
#include <texts/TextKeysAndLanguages.hpp>
#include <BitmapDatabase.hpp>
```

It is imperative that hardcoded magic numbers are not used to access assets used in TouchGFX, as the numerical values of these keys are subject to change often. If an access is attempted to an asset that doesn't exist, or no longer exists, it is treated as a hard fault and the display will reset.

12.3.2 Hard Fault – C++ Error Handling

The template setup supports writing helper functions in C++ as well as C. This means that you also have access to the C++ standard libraries. But with this must come some caution. Because this is a bare metal application, **try-catch error handling is not supported**, and any errors thrown by the C++ standard libraries will result in a hard fault and subsequent system reset.

12.3.3 Operating System - Task Mismanagement

Ensure that tasks are properly yielded at regular intervals. The watchdog is reset in its own independent OS task. If this task cannot be called at least once every 200ms, then the watchdog timer will hit 0 and result in a system reset. The watchdog task might not be called if a higher priority or equal priority task is not yielding itself to the scheduler. This can commonly happen in any created CAN task as improper use of the "*Dequeue_CanMessage*" function can result in said task not yielding properly.

If not a CAN related task, ensure that there is a reliable call of the "*osDelay*" function. This function is provided by CMSIS and if not available already, can be utilized by including the "*cmsis_os2.h*" header file.

