FOCUS LCDs
LCDs MADE SIMPLE®
Ph. 480-503-4295 | NOPP@FocusLCDs.com

TFT | OLED | GRAPHIC | CHARACTER | UWVD | SEGMENT | CUSTOM

Application Note FAN4224

Working with the E35RD-MW420-C

This application note will present the firmware to drive the E35RD-MW420-C TFT Display Module with an STM32H747I-DISCO microcontroller development board.

Contents

List of Figures

The E35RD-MW420-C Firmware Project

This application note will present the hardware and walk through developing the firmware required to drive the E35RD-MW420-C MIPI TFT Display with an STM32H747I-DISCO microcontroller board from ST Microelectronics. Driving a MIPI DSI display with a microcontroller presents a few challenges. The STM32H747 has the required bandwidth and I/O pins but lacks enough internal SRAM for a full frame buffer. The DISCO board presented has an external SDRAM chip used for implementing the display frame buffer.

Previous application notes (FAN4221, FAN4222, and FAN4223) had discussed the hardware required, an overview of the firmware, and how to modify the firmware from the E43RB1-FW405-C MIPI TFT Display for the E50RA-I-MW490-C MIPI TFT Display. FAN4224 develops the firmware for the E35RD-MW420-C while reusing the source files common to the MIPI displays.

Figure 1: 35RD-MW420-C, KAB-M20SD-01 Adapter, and STM32H7 Disco Running Demo

Introduction

Developing the firmware for the E35RD-MW420-C MIPI TFT Display will be made simpler by reusing the source files for the E43GB-I-FW405-C (provided by Focus LCDs upon request) and making similar modifications as presented in FAN4223. Like the previous application notes, it is assumed that an adapter PCB will be used to connect the display to the development board. Again, the adapter PCB can be designed by the end user, or the Focus LCDs prototyping adapter board can be used for the project.

Contact Focus LCDs for more information on the Adapter Board and firmware.

The E35RD-MW420-C display used in this application is a 3.5” TFT with a 480 x 800 RGB pixel resolution from Focus LCDs. This display is interfaced over a 2-lane MIPI DSI protocol with a 20-pin FPC cable. An ST Microelectronics STM32H747XI microcontroller, specifically the Discovery Development Board, will drive the display.

The main features of the E43RB1-FW405-C are:

- 3.5” Diagonal Display, 480 x 800 RGB Pixel Resolution
- Up to 65k/262k/16.7M (24-bit) Color
- 2-Lane MIPI DSI Interface with 20-pin FPC Cable
- Transmissive/Normally Black Display Mode
- White LED Backlight
- ST7701S Display Controller
- Capacitive Touch Panel () – Touch Mode: 5-Point and Gesture
- Typical Operating Voltage: 3.3V

Figure 2: E35RD-MW420-C 3.5" MIPI TFT Display

Hardware Requirements

The development board that will drive the display is an STM32H747I-DISCO Discovery Kit. The Kit comes with its own display but for this app note, the stock display will be replaced with the E35RD-MW420-C. See FAN4221 for more information.

Figure 3: STM32H747I-Disco Development Board

In this application, the Focus LCDs MIPI DSI Adapter Board v1.0 is used to adapt from the Q Strip connector to an FFC connector on the PCB. This adapter board is still in development. Contact Focus LCDs for more information.

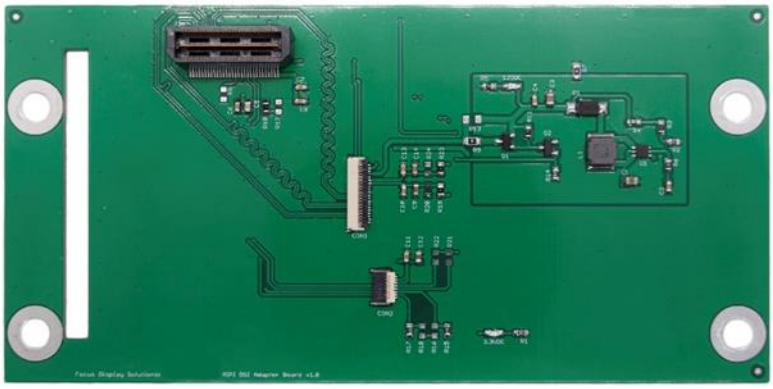

Figure 4: Focus LCDs MIPI DSI Adapter Board v1.0

Development Tools

STM32CubeIDE

Developing applications typically will use STM32CubeMX for setting up peripheral initialization and STM32CubeIDE (integrated development environment) to code the application side. These tools can be downloaded from the ST website. In this application, only the STM32CubeIDE will be used in the development of the firmware.

After installation, STM32CubeIDE might request additional downloads depending on the device series integrated into the project. When developing the firmware, an additional download for the H7 series of devices was required.

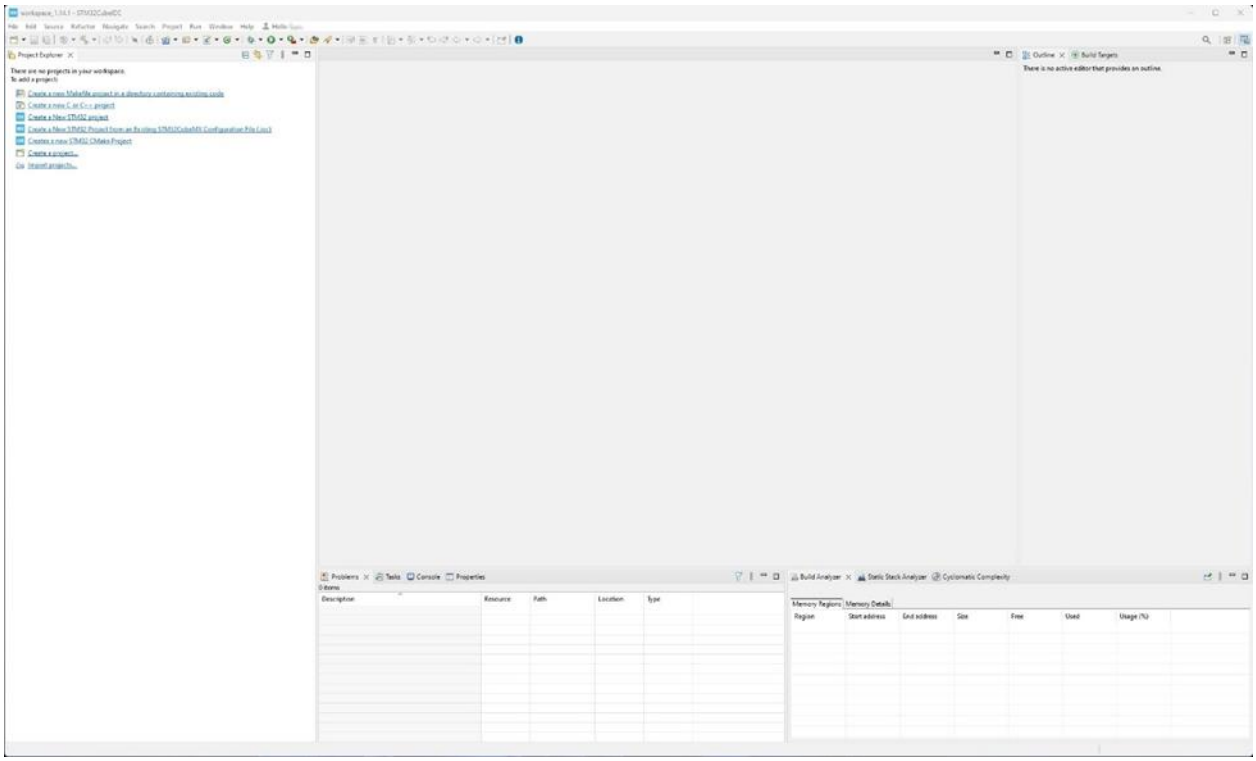

Figure 5: STM32CubeIDE Integrated Development Environment

Project Development

In this section, the initial project generation will be presented. The steps to create a new project, what files are to be added, initial modifications, and test building the firmware will be shown.

Start New Project

Open STM32CubeIDE and select “Create a New STM32 Project” from the available selections.

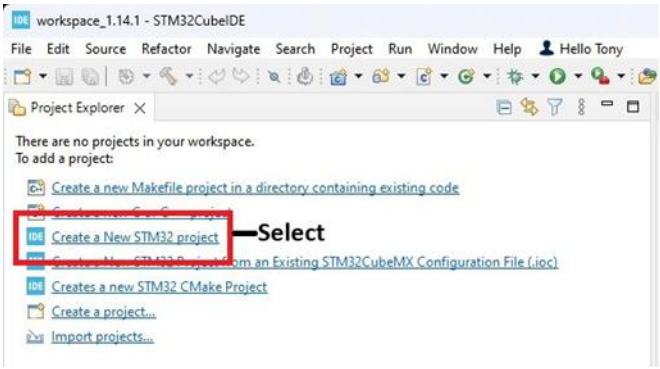

Figure 6: Start a New Project

The IDE will perform a short download, then open the product selection window.

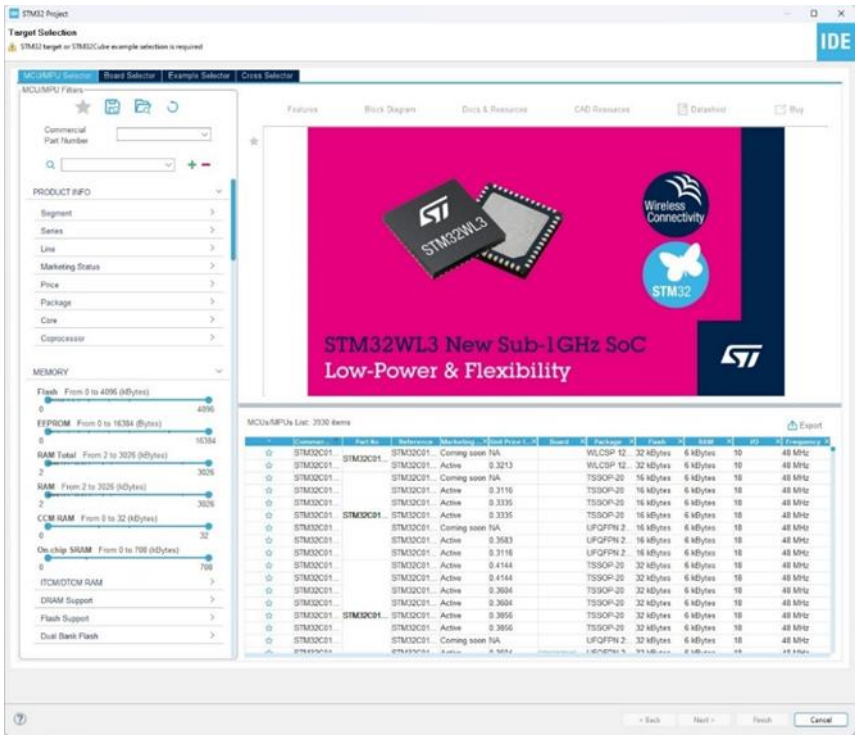

Figure 7: Target Selection Window

Select the MCU

In the “Target Selection” window, look for the box labeled “Commercial Part Number”. This is where the part number of the MCU on the STM32H747I-DISCO will be entered. Enter “STM32H747XI” and select the presented device.

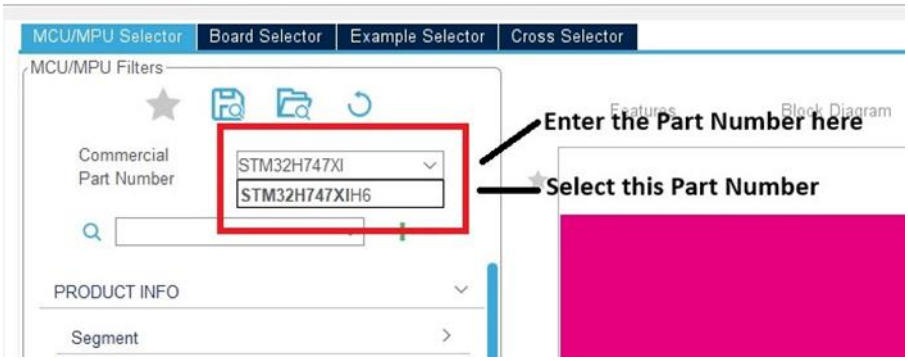

Figure 8: Select the STM32H747XIH6 Device

Once the MCU has been selected then click on the device in the “Features” section as outlined in blue.

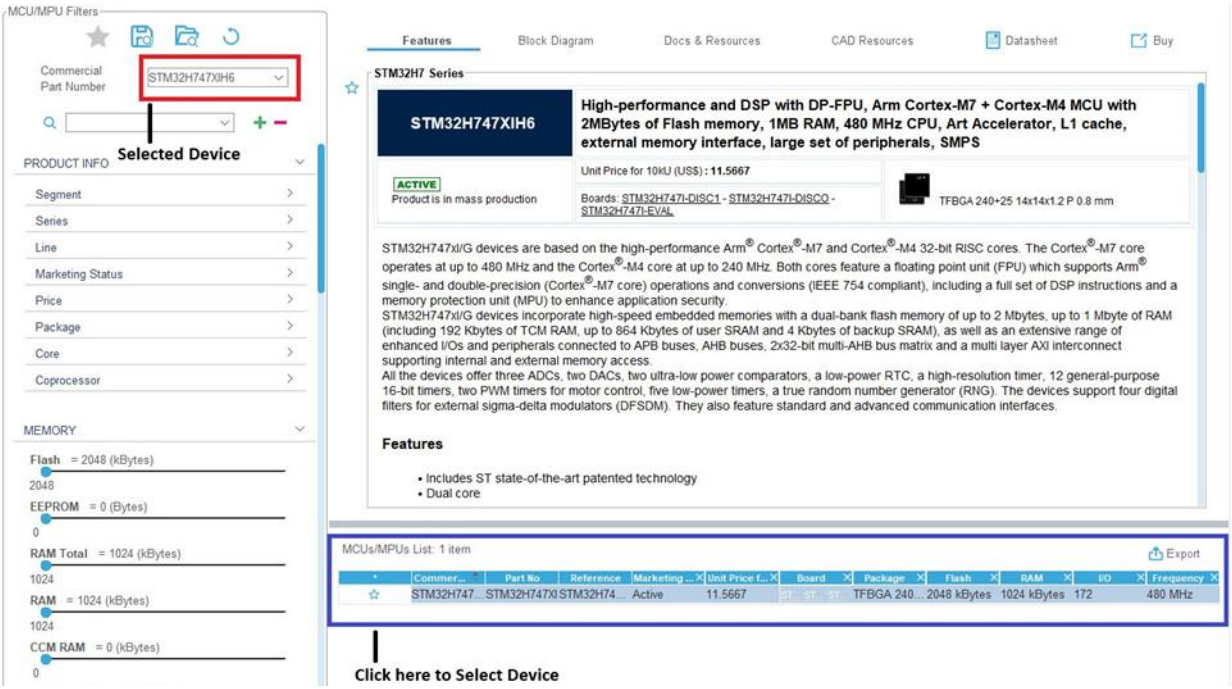

Figure 9: Selecting the STM32H747XIH6 Device in Features Section

Then click “Next” as highlighted in green.

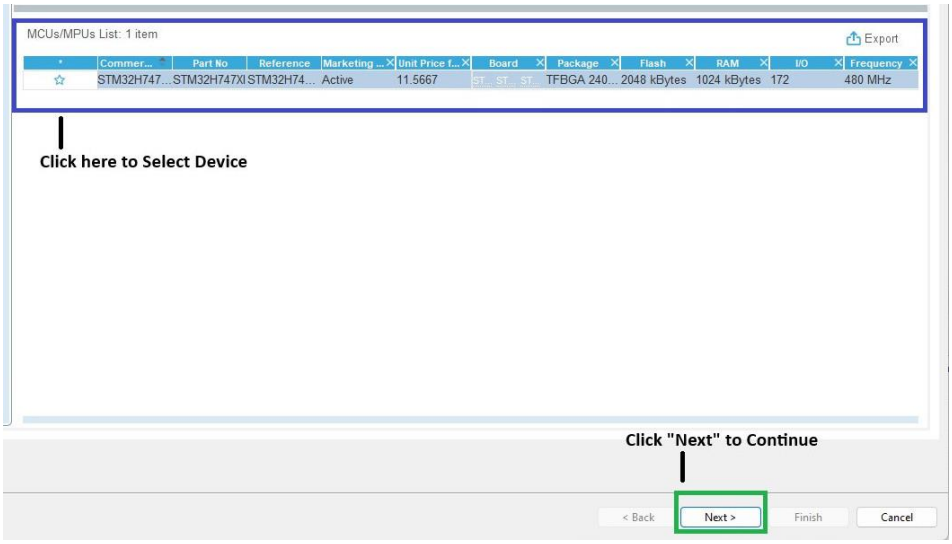

Figure 10: Select "Next" to Continue Project Creation

Name the Project

There are several steps left to finish the project creation. Naming the project is the next step. Keep the default settings as shown in the image below.

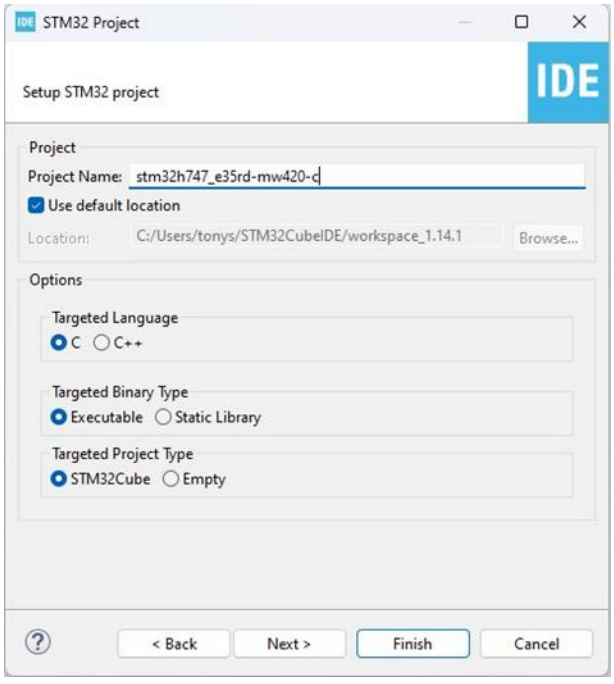

Figure 11: Naming the Project

After naming the project, select “Next” to verify the included libraries.

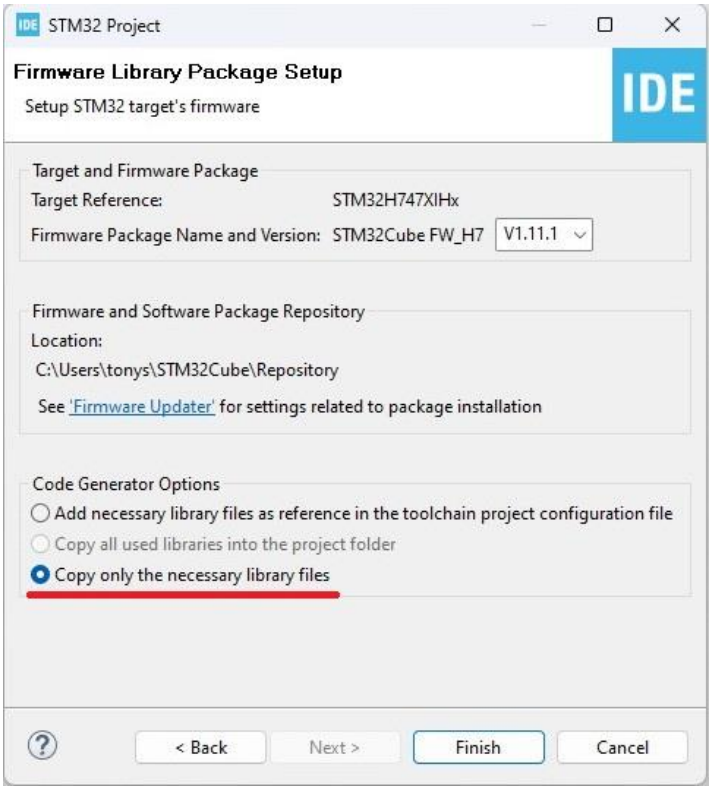

Figure 12: Library Package Setup

Verify that only the necessary libraries are included.

The next two pop up dialogs have optional selections. In both cases, “Yes” was selected. The first is whether the Pinout Configuration Perspective is opened after project generation is completed. “No” can be selected with no effect on the code base.

Figure 13: Open the Pin Configuration Perspective

The second pop up dialog will influence code generation. The dialog asks if it should enable the Memory Protection Unit (MPU) to optimize the Speculative Read access of the M7 core. This will add code for configuring the registers involved with the MPU and Speculative Read. This is a minor amount of code and should not affect the operation of the display. Again, selecting “No” is a viable

option but in the case of this application note “Yes” was selected and the resulting code compiled into the firmware.

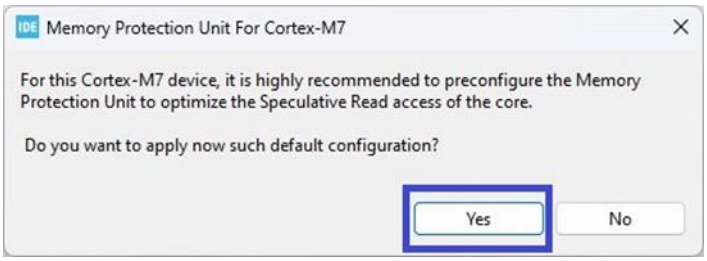

Figure 14: Memory Protection Unit Enable

Once those steps are completed the STM32CubeIDE will display:

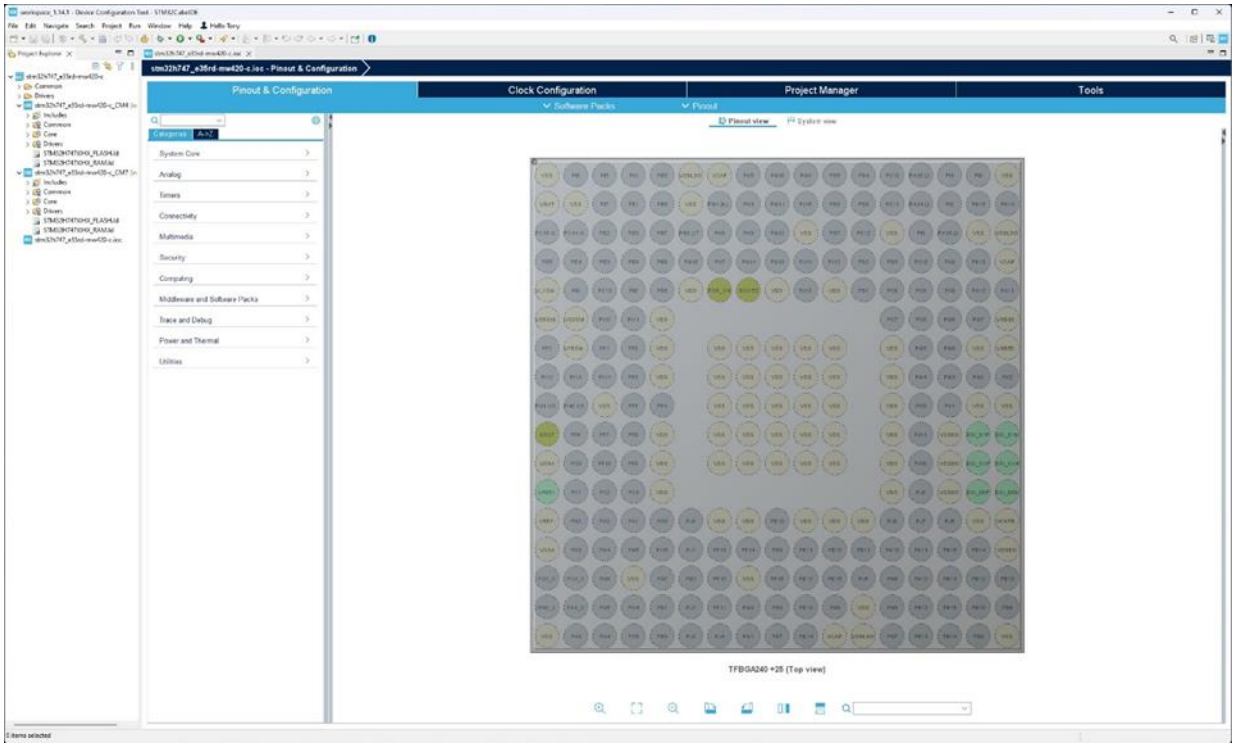

Figure 15: STM32CubeIDE Pin Perspective

Add the Files to the Project

The Pinout Configuration perspective can be closed, and the C/C++ perspective opened.

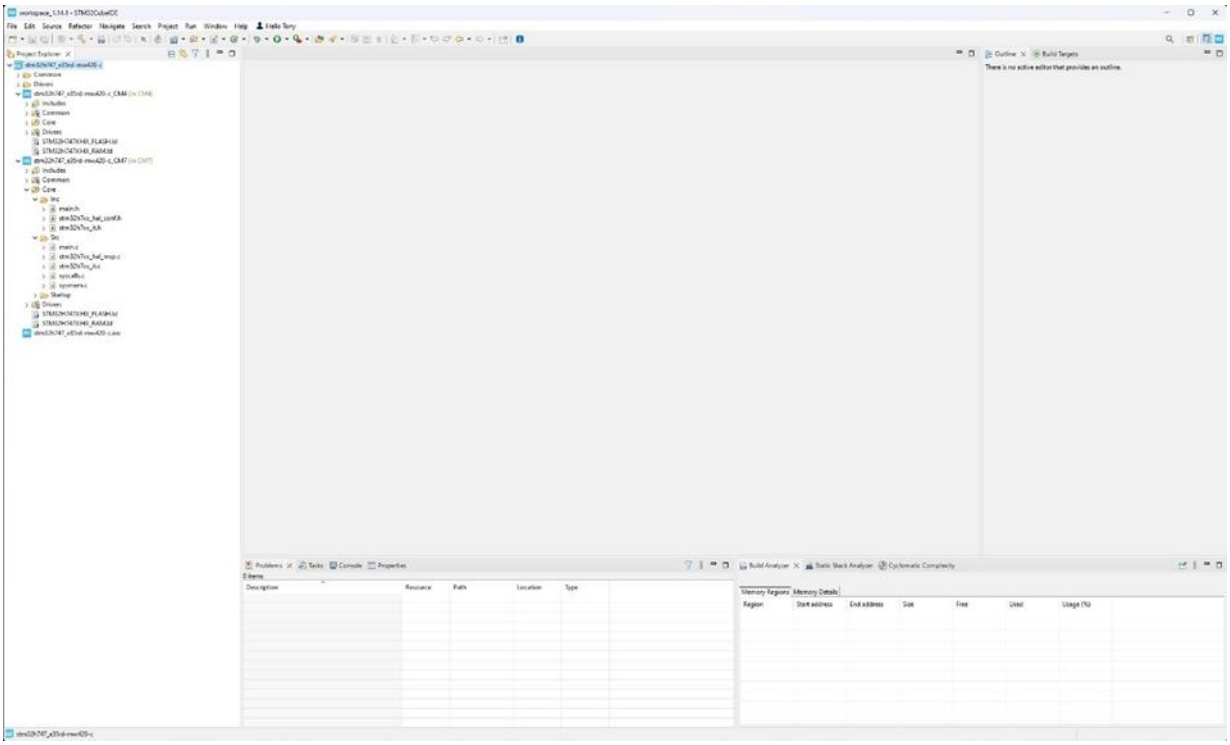

Figure 16: C/C++ Perspective

The next step in the creation of the project is to add in the header and source files provided by Focus LCDs and read FAN4222 and FAN4223. Many of the files used throughout these projects are the same. The only difference is the few files based on the differences in the display initialization and timing. Those modifications will be presented in the section: **Creating the Firmware Files for the E35RD-MW420-C**.

The complete project files for the **E35RD-MW420-C**, requiring no modification, can be provided by Focus LCDs upon request. This application note is meant as a guide to developing firmware for a display without completed project files.

First, add the include files to the project by first opening the development environment workspace and navigating to the CM7 Inc folder. As an example: **…/stm32h747_e35rd-mw420-c/CM7/core/Inc**. There will be 3 initial files, generated by the tools, in the folder.

Below is the list of header files to copy into the workspace Inc folder.

clock.h	debug.h	delay.h	dsi.h
fmc.h	focusimage.h	ft5x16.h	fwver.h
gui.h	i2cs.h	lcd.h	mcuid.h
misc.h	touch.h	tpc_descriptors.h	usart.h

Once the files are copied over to the proper workspace folder location, Project Explorer in the IDE will need to be refreshed to import the added files.

Right click on: **stm32h747_e35rd-mw420-c_CM7** to bring up the menu and select refresh. Once the refresh has been completed the newly added files will be part of the project.

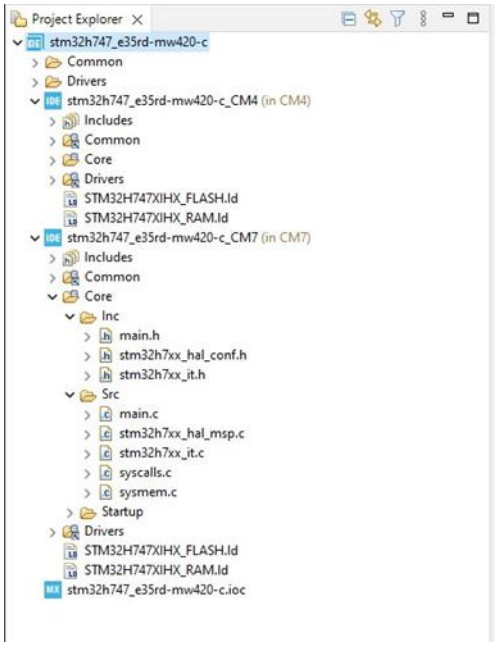

Figure 17: Project Explorer before Refresh

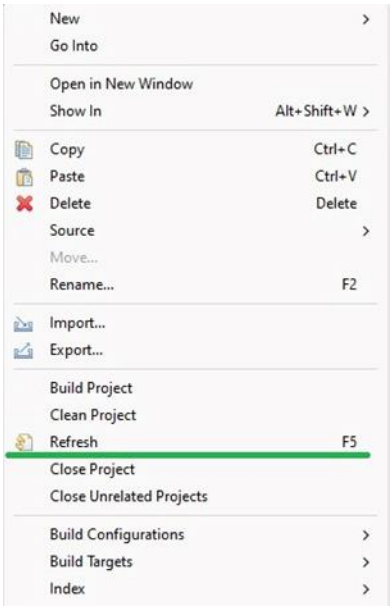

Figure 18: Right-click Menu: Refresh

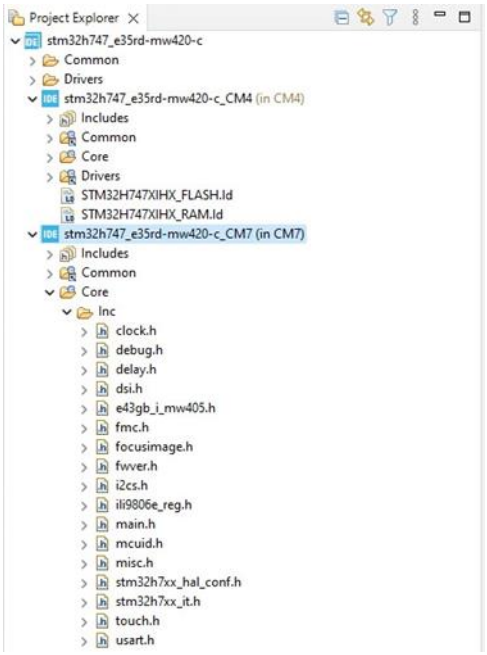

Figure 19: Refresh of Inc Folder

The same procedure will be used to add the necessary files to the source folder. The files will be placed in **…/workspace_1.14.1/stm32h747_e35rd-mw420-c/CM7/core/Src**. There are 5 files, generated by the dev tool, already in that folder. The files that will be added are.

clock.c	debug.c	delay.c	dsi.c
fmc.c	focusimage.c	ft5x16.c	fwver.c
gui.c	i2cs.c	lcd.c	misc.c
touch.h	usart.h		

Again, the project will need to be refreshed. Right click on: **stm32h747_e35rd-mw420-c_CM7** to bring up the menu and select refresh. Once the refresh has been completed the newly added files will be part of the project as shown below:

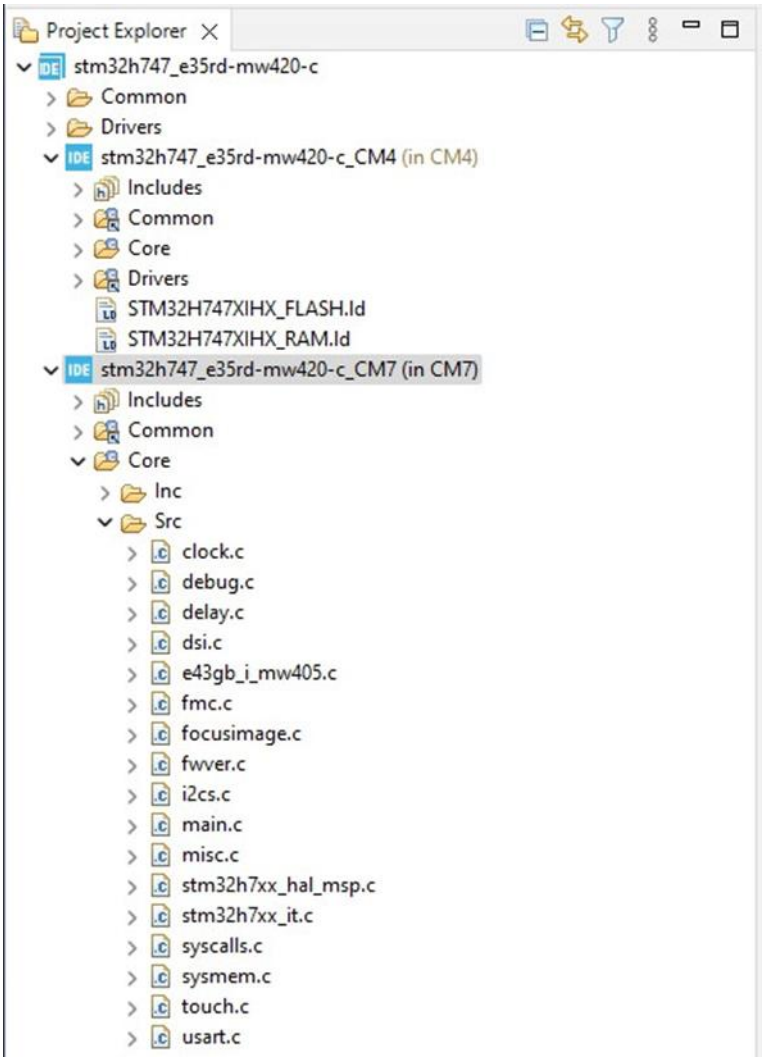

Figure 20: Refresh the Src Folder

Modify main.h and main.c

The header files must be included into the main.h file so that the resulting code can be compiled into the firmware. Insert the header files in the code section marked by: `/* USER CODE BEGIN Include */`. The debug macro can be placed in the section marked by: `/* USER CODE BEGIN EM */`. Modifications to main.h are shown in the image below:

```
75⊖/*****************************************************************************
76  * Includes
77  *****************************************************************************/
78 #include "stm32h7xx_hal.h"
79
80⊖/* Private includes ----------------------------------------------------------*/
81 /* USER CODE BEGIN Includes */
82 #include "mcuid.h"
83 #include "clock.h"
84 #include "delay.h"
85 #include "debug.h"
86
87 #include "fmc.h"
88 #include "dsi.h"
89
90 #include "touch.h"
91 #include "gui.h"
92
93 /* USER CODE END Includes */
94
95⊖/* Exported types ------------------------------------------------------------*/
96 /* USER CODE BEGIN ET */
97
98 /* USER CODE END ET */
99
100⊖/* Exported constants --------------------------------------------------------*/
101 /* USER CODE BEGIN EC */
102
103 /* USER CODE END EC */
104
105⊖/* Exported macro ------------------------------------------------------------*/
106 /* USER CODE BEGIN EM */
107 #define debugMAIN(type, ...)            printDEBUG(type, "SYS", __VA_ARGS__)
108 /* USER CODE END EM */
```

Figure 21: Main.h Modifications

Next, main.c will require some additions for the application code. Looking at the image below, in the first green boxed area, the peripheral and subsystem initialization functions were added under the section: `/* USER CODE BEGIN 2 */`. The first few lines of code enable User LED1 on the STM32H747 Discovery board. LED1 will blink at a 1Hz rate.

In the same section the macro **`initDEBUG()`** and **`debugMAIN()`** are added to allow debug messages to be sent over the USART (Universal Synchronous/Asynchronous Receiver Transmitter).

Finally, the **`chkGUI()`** function is called in the **`while(1)`** loop to respond to touch events. The chk4TimeoutSYSTIM() is used to create a 1 Hz timeout so that use LED1 can be blinked once a second. This verifies the application is running.

e35rd_mw420_c.h | e35rd_mw420_c.c | main.h | main.c

```
178   // Verify the MCU has started correctly by turning on LED1
179   RCC->AHB4ENR |= RCC_AHB4ENR_GPIOIEN;
180   GPIOI->MODER &= ~(GPIO_MODER_MODE12);
181   GPIOI->MODER |= (GPIO_MODER_MODE12_0);
182   GPIOI->ODR |= 0x1000;
183
184   // Enable the I and D cache
185   SCB_EnableICache();
186   SCB_EnableDCache();
187
188   initCLOCK();
189   initSYSTIM();
190
191   initDEBUG("[CM7]", '4', "STM32H747 MIPI Display Test");
192   initMICTIM();
193
194   initFMC();
195
196   // This calls initTouch() and initDSI()
197   initGUI();
198
199   debugMAIN(0, "init done\n");
200
201   uint32_t timer = getSYSTIM();
202   /* USER CODE END 2 */
203
204   /* Infinite loop */
205   /* USER CODE BEGIN WHILE */
206   while (1)
207   {
208     chkGUI();
209
210     if(chk4TimeoutSYSTIM(timer, 1000)==(SYSTIM_TIMEOUT))
211     {
212         timer = getSYSTIM();
213         GPIOI->ODR ^= 0x1000;
214     }
215     /* USER CODE END WHILE */
216
217     /* USER CODE BEGIN 3 */
218   }
```

Figure 22: Main.c Modifications

Change Optimization Settings

The next step in creating the firmware is setting the optimization level. The typical setting is '-O2' for a good compromise between execution speed and code size. If left at '-O0' there would be no optimization for execution speed and code size.

First, in Project Explorer right click on the (CM7) project and select properties.

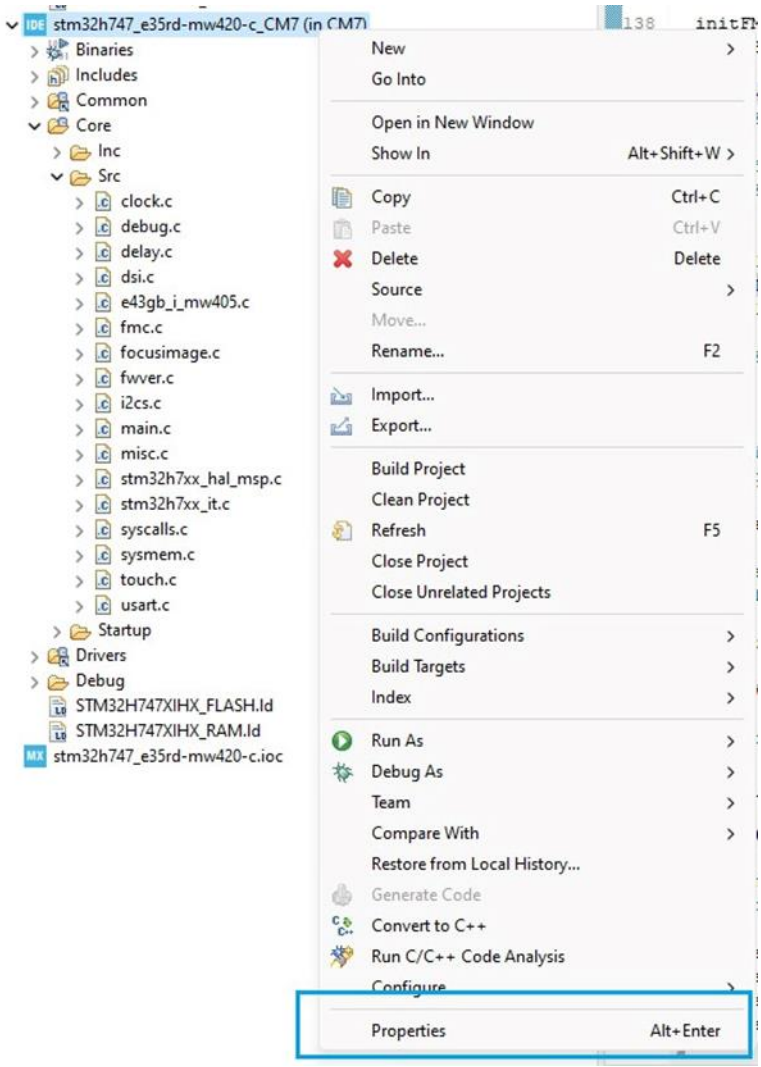

Figure 23:Project Properties Main Menu Item

In the pop-up window, select C/C++ Build->Settings.

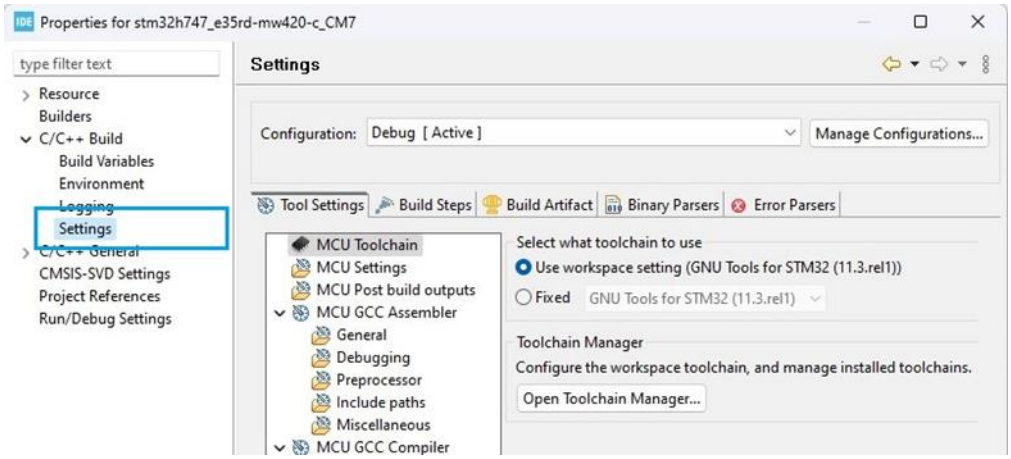

Figure 24: C/C++ Build Settings

Then in the Tool Settings tab on the right, select Optimization and set the Optimization level to -O2.

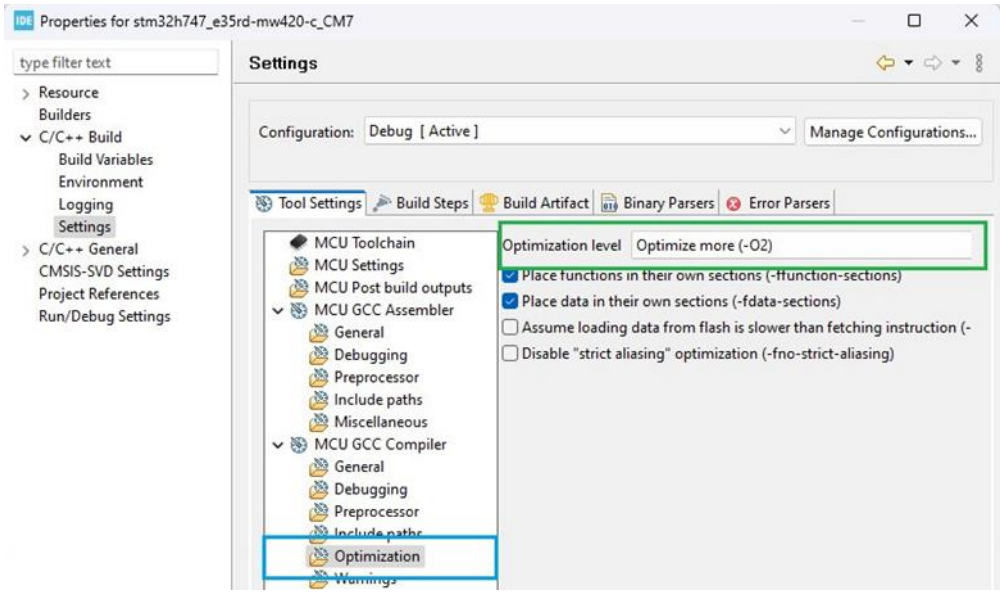

Figure 25: Optimization Level

Build the Project

The final step to setting up the project is to build the project. This step ensures that there are no missing dependencies. If there are any errors or warnings, resolve those issues before continuing with the firmware project.

Creating the Firmware Files for the E35RD-MW420-C

Using the E43GB-I-MW405-C firmware files as a reference, the E35RD-MW420-C display will be discussed in this section. This will guide the end user through the necessary steps of creating the firmware for the display.

These are the basic steps to creating the firmware:

1. Create the display specific files – e35rd_mw420_c.c/.h.
2. Edit the header file with the inclusion guard, include headers, and API
3. Create the display controller header file – st7701s.h.
4. Edit the header file to add the **`#define`** for the ST7701S registers.
5. Edit the e35rd_mw420_c.c file with the **`#include`** header file names.
6. Check the LCD controller chip datasheet, the Focus LCDs initialization code, and continue to modify the source file as necessary.
7. Edit the .c file with the new display parameters.
8. Make the initialization sequence changes to **`runInitSeqLCDConfig()`** function.

These steps will be discussed in the following sections.

Create the E35RD-MW420-C Header and Source Files

Using the e43gb_i_mw405_c.h and .c files as examples, create a new header file in the Inc folder and a new source file in the Src folder. Name the files e35rd_mw420_c.h and e35rd_mw420_c.c. Once that is completed, code will be added to the new header file.

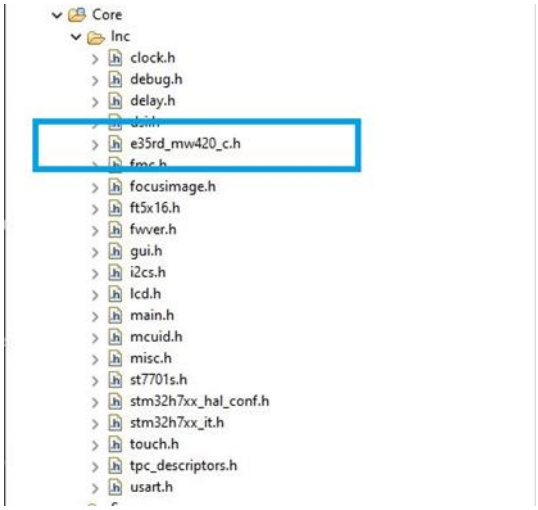

Figure 26: New Header File

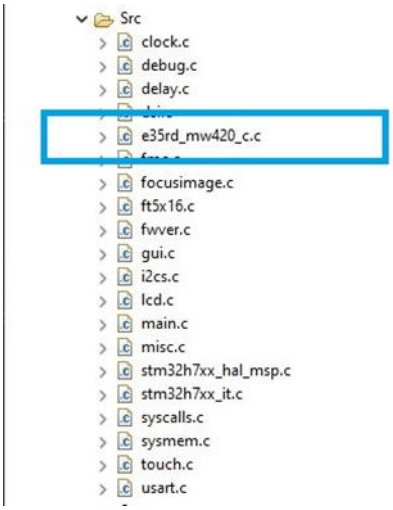

Figure 27: New Source File

Editing the Header File

In the header file, verify that the **`#ifndef E35RD_MW420_C_H`** matches the new header file name.

```
#ifndef E35RD_MW420_C_H
#define E35RD_MW420_C_H
```

Figure 28: Include Guard

Add the include preprocessor commands for the standard C library headers and for the project headers. The **`<stdarg>`** header is needed for the **`__VA__ARGS__`** used in the debug code inside the source file.

```
88 /*****************************************
89  * Includes
90  *****************************************
91 #include <stdint.h>
92 #include <stdarg.h>
93 #include "mcuid.h"
94 #include "debug.h"
95 #include "delay.h"
96 #include "dsi.h"
97 #include "st7701s.h"
```

Figure 29: Include Headers

At this point defines could be added for the LCD width and height in pixels to help make the code more portable between displays. Once that is complete the function prototypes for the API can be placed in the file.

```
120 /*********************************************
121  * API
122  *********************************************
123 void initLCDConfig(LCDCONF * lcdconf);
124 void runInitSeqLCDConfig(LCDCONF * lcdconf);
```

Figure 30: API

Create and Edit the ST7701S Header File

In the header file, verify that the **`#ifndef ST7701S_REG_H`** matches the new header file name.

```
65 #ifndef ST7701S_REG_H
66 #define ST7701S_REG_H
```

Figure 31: ST7701S Header File Include Guard

Looking at the ST7701S datasheet, the section on the commands/registers is what will populate the header file. Start by #define the commands and data bytes for changing register pages.

```
e35rd_mw420_c.h  e35rd_mw420_c.c  main.h  main.c  st7701s.h
77 /*****************************************************************************
78  * Global Constants/Defines
79  *****************************************************************************/
80 /***   Enable Extended Command Registers - has 5 parameters   ***************/
81 #define ST7701S_CN2BKSEL         0xFF    // Enable Command Bank Select register
82 #define ST7701S_CN2BKSEL_PARAM1  0x77    // Parameter 1 - required
83 #define ST7701S_CN2BKSEL_PARAM2  0x01    // Parameter 2 - required
84 #define ST7701S_CN2BKSEL_PARAM3  0x00    // Parameter 3 - required
85 #define ST7701S_CN2BKSEL_PARAM4  0x00    // Parameter 4 - required
86                                          // Parameter 5 - use only 1 option below
87 #define ST7701S_CMD1             0x00    // Command 1 register set
88 #define ST7701S_CMD2_BNK0        0x10    // Command 2 Bank 0 register set
89 #define ST7701S_CMD2_BNK1        0x11    // Command 2 Bank 1 register set
90 #define ST7701S_CMD2_BNK3        0x13    // Command 2 Bank 3 register set
```

Figure 32: Page/Bank Change Command and Data

Next, add the rest of the command registers to the header file.

```
e35rd_mw420_c.h  e35rd_mw420_c.c  main.h  main.c  st7701s.h
93 /*************   Command 1 Bank 0   ***************************************/
94 #define ST7701S_NOP              0x00    // no operation
95 #define ST7701S_SWRESET          0x01    // software reset
96 #define ST7701S_RDDID            0x04    // read device ID (returns 3 bytes)
97 #define ST7701S_RDNUMED          0x05    // read # of DSI errors
98 #define ST7701S_RDRED            0x06    // read first pixel of color R
99 #define ST7701S_RDGREEN          0x07    // read first pixel of color G
100 #define ST7701S_RDBLUE          0x08    // read first pixel of color B
101 #define ST7701S_RDDPM           0x0A    // read display power mode
102 #define ST7701S_RDDMADCTL       0x0B    // read display MADCTL
```

Figure 33: Register Defines

Some command registers have named bit fields. Define these bit names so they can later be used to configure their associated register. See the COLMOD register and bit defines below for an example.

```
e35rd_mw420_c.h  e35rd_mw420_c.c  main.h  main.c  st7701s.h
140 /*************   Command 1 COLMOD   ********************************************/
141 #define ST7701S_COLMOD              0x3A    // interface pixel format
142 #define BPP16                       0x50    // 16 bits per pixel format: 65k color
143 #define BPP18                       0x60    // 18 bits per pixel format: 262k color
144 #define BPP24                       0x70    // 24 bits per pixel format: 16.7M color
```

Figure 34: Bit Defines

Complete all the register and bit defines for the ST7701S.

Editing the Source File

First, looking at the datasheet for the E35RD-MW420-C, notice the TFT controller chip is the ST7701S.

In the e35rd_mw420_c.c source file, the #include directives must be added for E35RD-MW420-C and the ST7701S. The header file include should be **`#include "e35rd_mw420_c.h"`** and **`#include "st7701s"`**.

The debugging system requires a constant module name and a macro that provides an alias to the function pointer for printing debug messages.

```
95 static const uint8_t moduleName[] = "LCDConfig";
96
97 #define debugLCDCONF(type, ...)     printDEBUG(type, (char *)moduleName, __VA_ARGS__)
```

Figure 35: Constant Name and Macro Alias

Setting up Arrays for the Command Paramters

Many of the commands for controlling the initialization of the display have associated parameters. One of the first items to code are the arrays with the data for the commands as the DSI packet write functions (provided by the ST include and source files generated with project creation) require arrays when there is more than 1 byte of data for a command.

Different display controllers have command sequences to access additional register pages or banks.

```
105 /*****************************************************************************
106  * Private Variables
107  *****************************************************************************
108 // Page/Bank change to access the registers on different pages/banks
109 // 0x77, 0x01, 0x00, 0x00, 0x00
110 uint8_t Page0[] = {ST7701S_CN2BKSEL_PARAM1, ST7701S_CN2BKSEL_PARAM2,
111                    ST7701S_CN2BKSEL_PARAM3, ST7701S_CN2BKSEL_PARAM4,
112                    ST7701S_CMD1};
113 // 0x77, 0x01, 0x00, 0x00, 0x10
114 uint8_t Page1[] = {ST7701S_CN2BKSEL_PARAM1, ST7701S_CN2BKSEL_PARAM2,
115                    ST7701S_CN2BKSEL_PARAM3, ST7701S_CN2BKSEL_PARAM4,
116                    ST7701S_CMD2_BNK0};
117 // 0x77, 0x01, 0x00, 0x00, 0x11
118 uint8_t Page2[] = {ST7701S_CN2BKSEL_PARAM1, ST7701S_CN2BKSEL_PARAM2,
119                    ST7701S_CN2BKSEL_PARAM3, ST7701S_CN2BKSEL_PARAM4,
120                    ST7701S_CMD2_BNK1};
121 // 0x77, 0x01, 0x00, 0x00, 0x13
122 uint8_t Page3[] = {ST7701S_CN2BKSEL_PARAM1, ST7701S_CN2BKSEL_PARAM2,
123                    ST7701S_CN2BKSEL_PARAM3, ST7701S_CN2BKSEL_PARAM4,
124                    ST7701S_CMD2_BNK3};
```

Figure 36: Arrays for Command Pages

Initial Parameter Configuration

The parameters of the TFT LCD will be set and initialized in the `void` **`initLCDConfig`**`(LCDCONF *`**`lcdconfig`**`)` function in the e35rd_mw420_c.c file. The argument **LCDCONF** is a structure containing all the essential information required by the DSI controller to configure the MIPI DSI interface.

Command or Video Mode and Additional Configuration

Identifying whether the display operates in MIPI video or command mode can be verified by reviewing the controller datasheet for GRAM. If the controller has GRAM, it can operate in command mode but if there is no GRAM it must operate in video mode. Reviewing the ST7701S datasheet shows that it has no internal GRAM and operates in video mode. In the source file the mode is set to **DSI_VIDEO_MODE** and does not need to be changed as both displays use the same controller.

```
129  *****************************************************************************/
130 void initLCDConfig(LCDCONF* lcdconf)
131 {
132
133     // set initial mode to video
134     lcdconf->mode = DSI_MODE_VIDEO;
```

Figure 37: DSI Video Mode Setting

Looking at the datasheet, the display resolution is 480 x 800. Modify the configuration height and width. The **LCD_WIDTH** and **LCD_HEIGHT** are defined in the e35rd_mw420_c.h file.

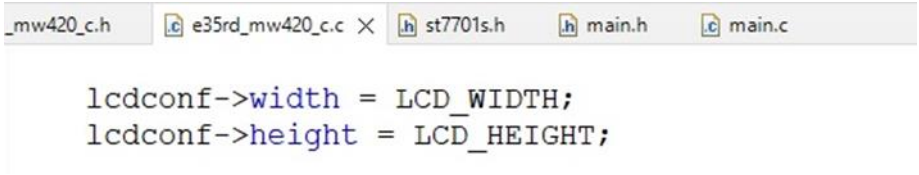

Figure 38: Adjusted Height and Width

The next set of parameters to consider are the display timings. These can sometimes be found in the display datasheet or from the comments section of the display initialization file. Both can be found on the Focus LCDs website or are available upon request. Carry the over the display timings to the LCDCONF structure.

```
NOTE:VCI=3.3V,IOVCC=1.8V,
Display resolution:480*800
params->dsi.vertical_sync_active=2
params->dsi.vertical_backporch=20
params->dsi.vertical_frontporch=10
params->dsi.horizontal_sync_active=2
params->dsi.horizontal_backporch=60
params->dsi.horizontal_frontporch=10
params->MIPI_CLOCK=(350)Mbps
params->dsi.PLL_CLOCK=(175)Mbps //FOR MTK
params->RGB_CLOCK=(28)MHZ
Frame Rate=60HZ
```

Figure 39: E35RD-MW420-C Display Timings from the Display Initialization File

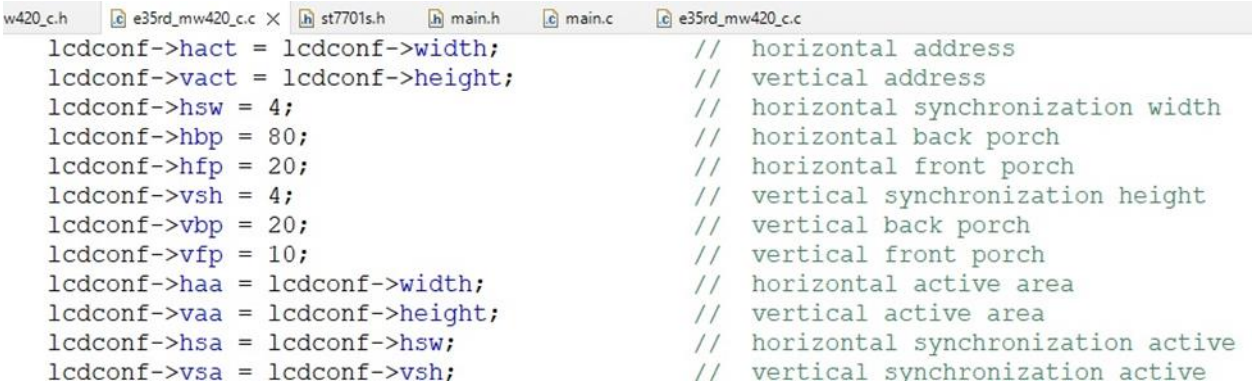

Figure 40: Display Timings

Due to slight variations in timing, some parameters may need to be adjusted to ensure proper functioning of the display.

Configuring the Initialization Sequence

Having adjusted the timings for the display, the initialization sequence will be modified for the new display. In the previous section, the display timings were retrieved from the display initialization file. The commands for display initialization are included in that file.

```
//*****************************/
write_command(0xFF);
write_data(0x77);
write_data(0x01);
write_data(0x00);
write_data(0x00);
write_data(0x13);
```

Figure 41: Switch Command Pages

There are two types of writes, command and data. The **`write_command()`** is the address of the register where the data will be written. The **`write_data()`** is the data that will be written to the register.

Commands that take more than 1 parameter require the use of the long write function, example above **`longWriteDSI(lcdconf->virch, long_pkt_write_type, 5, 0xFF, Page3)`**. Commands that require 1 or no data parameters are call short write functions like **`shortWriteDSI(lcdconf->virch, short_pkt_write_type, 0xEF, 0x08)`**. In the initialization code when a command has no corresponding data then a short write is used with the last parameter being set to **`0x00`**.

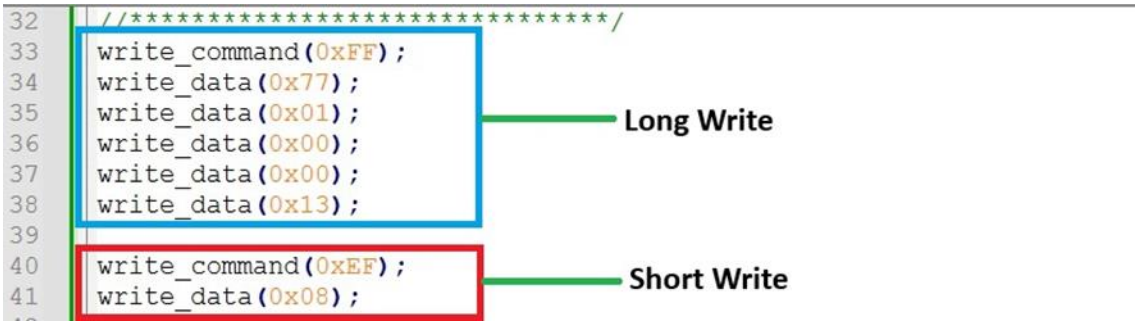

Figure 42: Display Initialization Long and Short Writes

The display initialization code is generic 'C' code to show the concept of registers and data to be placed in those registers. This is where the MIPI DSI concepts of short and long write packets are used for data transfer. The end user needs to convert the display initialization code into short and long DSI packet writes. The function used for calling all the initialization code is

```
mw420_c.h | e35rd_mw420_c.c | st7701s.h | main.h | main.c

void runInitSeqLCDConfig(LCDCONF* lcdconf)
{
```

Figure 43: runInitSeqLCDConfig() Function

Continue to edit the `void runInitSeqLCDConfig(LCDCONF *lcdconf)` function. This is required for all the write commands and associated data in the display initialization file. Shown below is a section of the additions to **`runInitSeqLCDConfig()`** function.

```
20_c.h | e35rd_mw420_c.c | st7701s.h | main.h | main.c | e35rd_mw420_c.c
  shortWriteDSI(lcdconf->virch, short_pkt_write_type, 0xB0, 0x5D);    // Vop
  shortWriteDSI(lcdconf->virch, short_pkt_write_type, 0xB1, 0x6D);    // VCOM = 5A
  shortWriteDSI(lcdconf->virch, short_pkt_write_type, 0xB2, 0x87);    // VGH = 15V
  shortWriteDSI(lcdconf->virch, short_pkt_write_type, 0xB3, 0x80);    // Test Command setting
  shortWriteDSI(lcdconf->virch, short_pkt_write_type, 0xB5, 0x49);    // VGL = -12V
  shortWriteDSI(lcdconf->virch, short_pkt_write_type, 0xB7, 0x85);    // Power Control 1
```

Figure 44: A Sample of the runInitSeqLCDConfig() Function

In the figure above, the **`shortWriteDSI()`** functions are used to send commands to the display. The short packet functions are used when a command with no data or 1 data parameter needs to be transmitted to the display. Long packet writes are used when there is more than 1 data parameter to transmit.

Looking at the display initialization code, there are 5 parameters to the page change command. In the figure below, an example of a long write packet is the DSI version of the page change command. The last parameter, **Page3**, is the array which stores the **5** parameters.

```
0_c.h | e35rd_mw420_c.c | st7701s.h | main.h | main.c | e35rd_mw420_c.c
 // Switch to Page 3
 longWriteDSI(lcdconf->virch, long_pkt_write_type, 5, 0xFF, Page3);
```

Figure 45: Page Switch Long Write Command

The commands that have several data parameters should have arrays for the data parameters to be passed to the **`longWriteDSI()`** function. Some of these commands include the Positive and Negative Gamma settings and the GIP settings. Once all the changes have been made to the various headers and source files, the demo will build and can be downloaded onto the STM32H7 Discovery board with the appropriate display attached.

Figure 46: E35RD-MW420-C Running the Focus LCDs Demo

Additional Comments

The source code with the modifications already completed with the addition of the required code for driving the touch panel can be provided by Focus LCDs upon request.

Summary

In Part 1, the hardware requirements for a MIPI display demo were presented. How to assemble the hardware was shown. Finally, a brief overview of the MIPI DSI interface and the STM32 DSI Host peripheral were discussed.

In Part 2 presented here, the development tools and frame buffer consideration were briefly discussed. The demonstration firmware operation was mentioned along with the touch interface. The following sections went through the operation of modifying the firmware for the E50RA-I-MW490-C display. Through the modification of the firmware, the structure and basic layout of the code is shown.

LCD Handling Precautions

- Do not store the TFT-LCD module in direct sunlight, best stored in a dark place
- Do not leave it exposed to high temperature and high humidity for a long period of time
- Recommended temperature range is 0 to 35 °C, relative humidity should be less than 70%
- Stored modules away from condensation as formation of dewdrops may cause an abnormal operation or failure of the module.
- Protect the module from static discharge
- Do not press or scratch the surface and protect it from physical shock or any force

Disclaimer

Buyers and others who are developing systems that incorporate FocusLCDs products (collectively, “Designers”) understand and agree that Designers remain responsible for using their independent analysis, evaluation, and judgment in designing their applications and that Designers have full and exclusive responsibility to assure the safety of Designers' applications and compliance of their applications (and of all FocusLCDs products used in or for Designers’ applications) with all applicable regulations, laws, and other applicable requirements.

Designer represents that, with respect to their applications, Designer has all the necessary expertise to create and implement safeguards that:

(1) anticipate dangerous consequences of failures

(2) monitor failures and their consequences, and

(3) lessen the likelihood of failures that might cause harm and take appropriate actions.

The designer agrees that prior to using or distributing any applications that include FocusLCDs products, the Designer will thoroughly test such applications and the functionality of such FocusLCDs products as used in such applications.

Revision History

Revision	Notes	Date
1.0.0	Initial Version	8/27/2024
1.1.0	Added Additional Code Sections for Clarification	9/6/2024
1.2.0	Added Running Demo Images	9/8/2024