

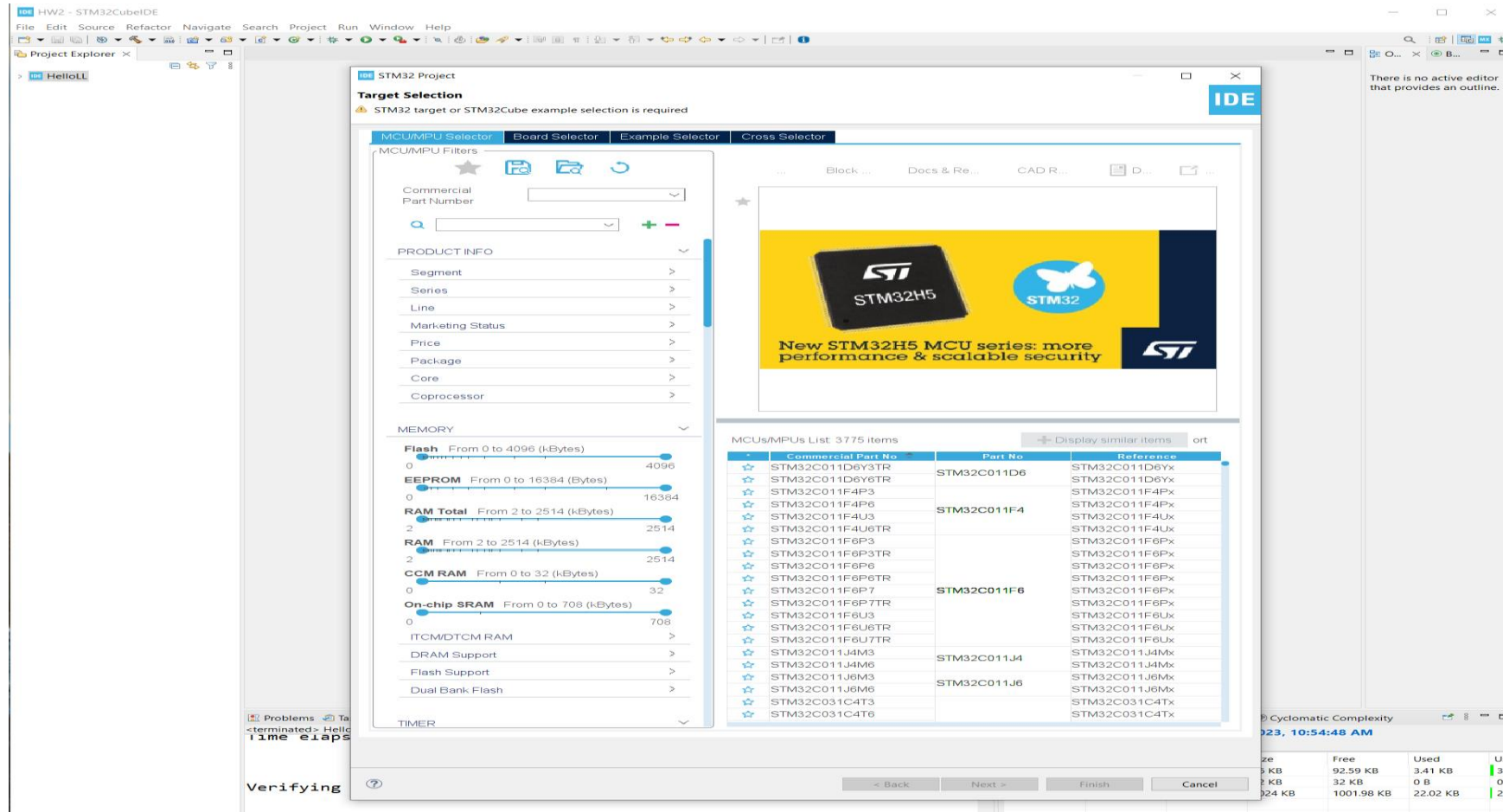
UCSD Embedded C Assignment 4

By

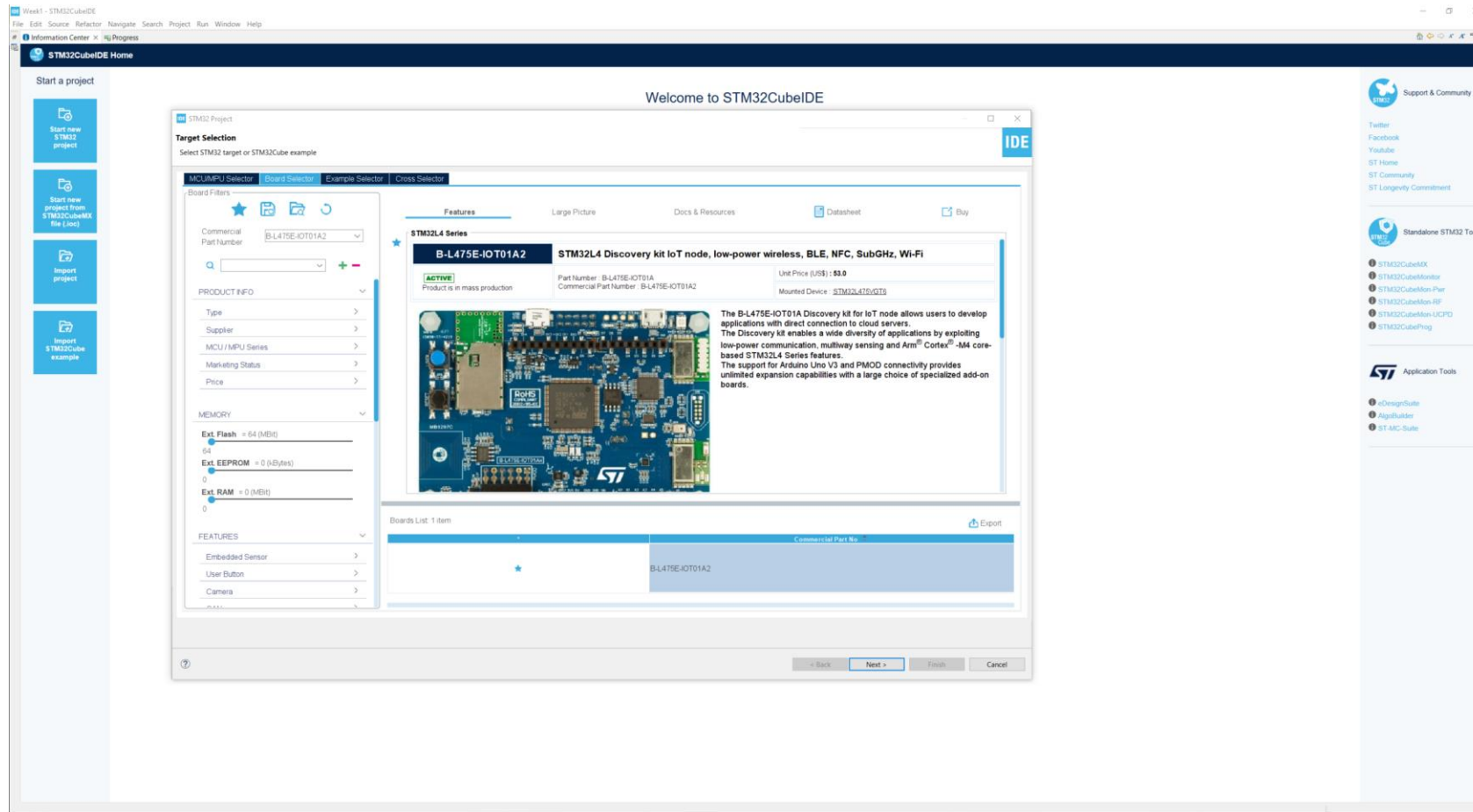
Hsuankai Chang

hsuankac@umich.edu

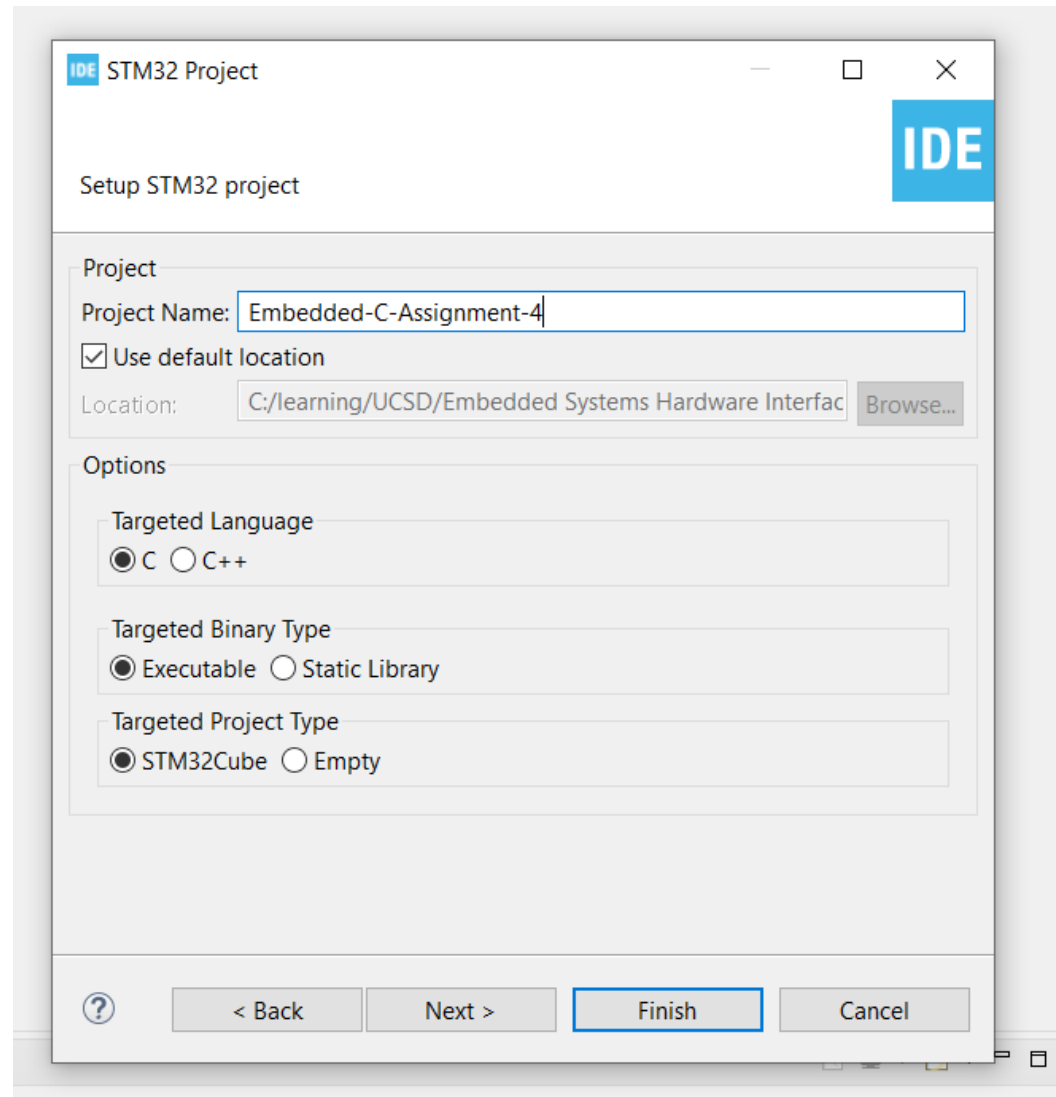
Step 1. Startup STM32CubeIDE and create new STM32 project



Step 2. Access board selector and type in the board you use, click Next



Step 3. Enter the project name then click Next



The image shows a 'Setup STM32 project' dialog box from the IDE. The window title is 'IDE STM32 Project'. The main heading is 'Setup STM32 project'. Under the 'Project' section, the 'Project Name' field contains 'Embedded-C-Assignment-4'. The 'Use default location' checkbox is checked. The 'Location' field shows 'C:/learning/UCSD/Embedded Systems Hardware Interfac' with a 'Browse...' button. Under the 'Options' section, 'Targeted Language' has 'C' selected, 'Targeted Binary Type' has 'Executable' selected, and 'Targeted Project Type' has 'STM32Cube' selected. At the bottom, there are buttons for '?', '< Back', 'Next >', 'Finish' (highlighted with a blue border), and 'Cancel'.

IDE STM32 Project

Setup STM32 project

Project

Project Name: Embedded-C-Assignment-4

☒ Use default location

Location: C:/learning/UCSD/Embedded Systems Hardware Interfac Browse...

Options

Targeted Language

☒ C ☐ C++

Targeted Binary Type

☒ Executable ☐ Static Library

Targeted Project Type

☒ STM32Cube ☐ Empty

? < Back Next > Finish Cancel

Step 4. See the firmware package name and version



The image shows a Windows-style dialog box titled "STM32 Project" with a subtitle "Firmware Library Package Setup". The subtitle also includes the instruction "Setup STM32 target's firmware". The dialog is divided into three sections: "Target and Firmware Package", "Firmware and Software Package Repository", and "Code Generator Options". In the first section, the "Target Reference" is "B-L475E-IOT01A2" and the "Firmware Package Name and Version" is "STM32Cube FW_L4 V1.17.2", with the version part highlighted by a blue selection box. The second section shows the "Location" as "C:\Users\hsuankai.chang\STM32Cube\Repository" and includes a link to the "Firmware Updater". The third section contains three radio button options for code generation, with the last option, "Copy only the necessary library files", being selected. At the bottom, there are buttons for "?", "< Back", "Next >", "Finish" (which is highlighted with a blue border), and "Cancel".

IDE STM32 Project

Firmware Library Package Setup

Setup STM32 target's firmware

Target and Firmware Package

Target Reference: B-L475E-IOT01A2

Firmware Package Name and Version: STM32Cube FW_L4 V1.17.2

Firmware and Software Package Repository

Location:
C:\Users\hsuankai.chang\STM32Cube\Repository

See ['Firmware Updater'](#) for settings related to package installation

Code Generator Options

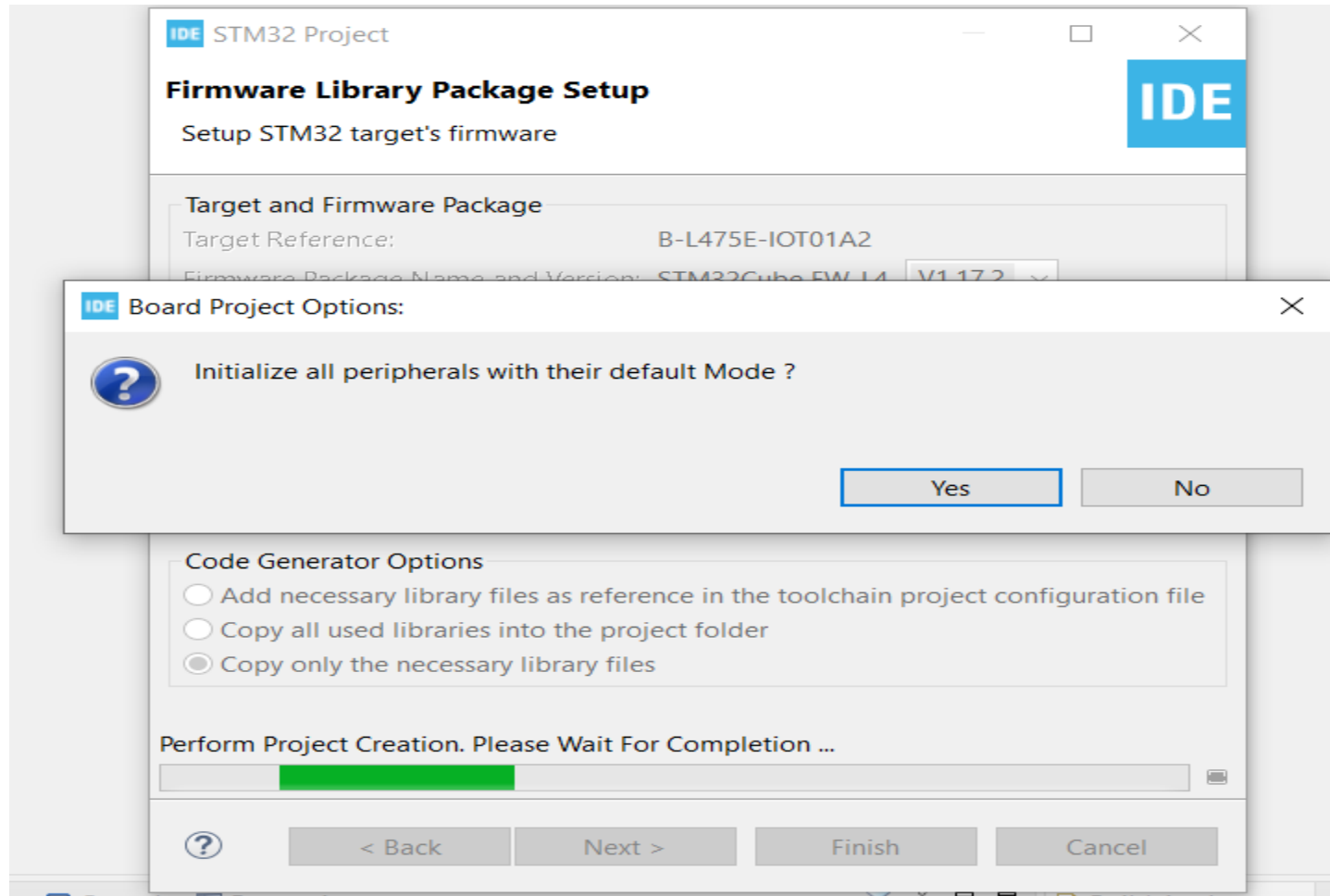
☐ Add necessary library files as reference in the toolchain project configuration file

☐ Copy all used libraries into the project folder

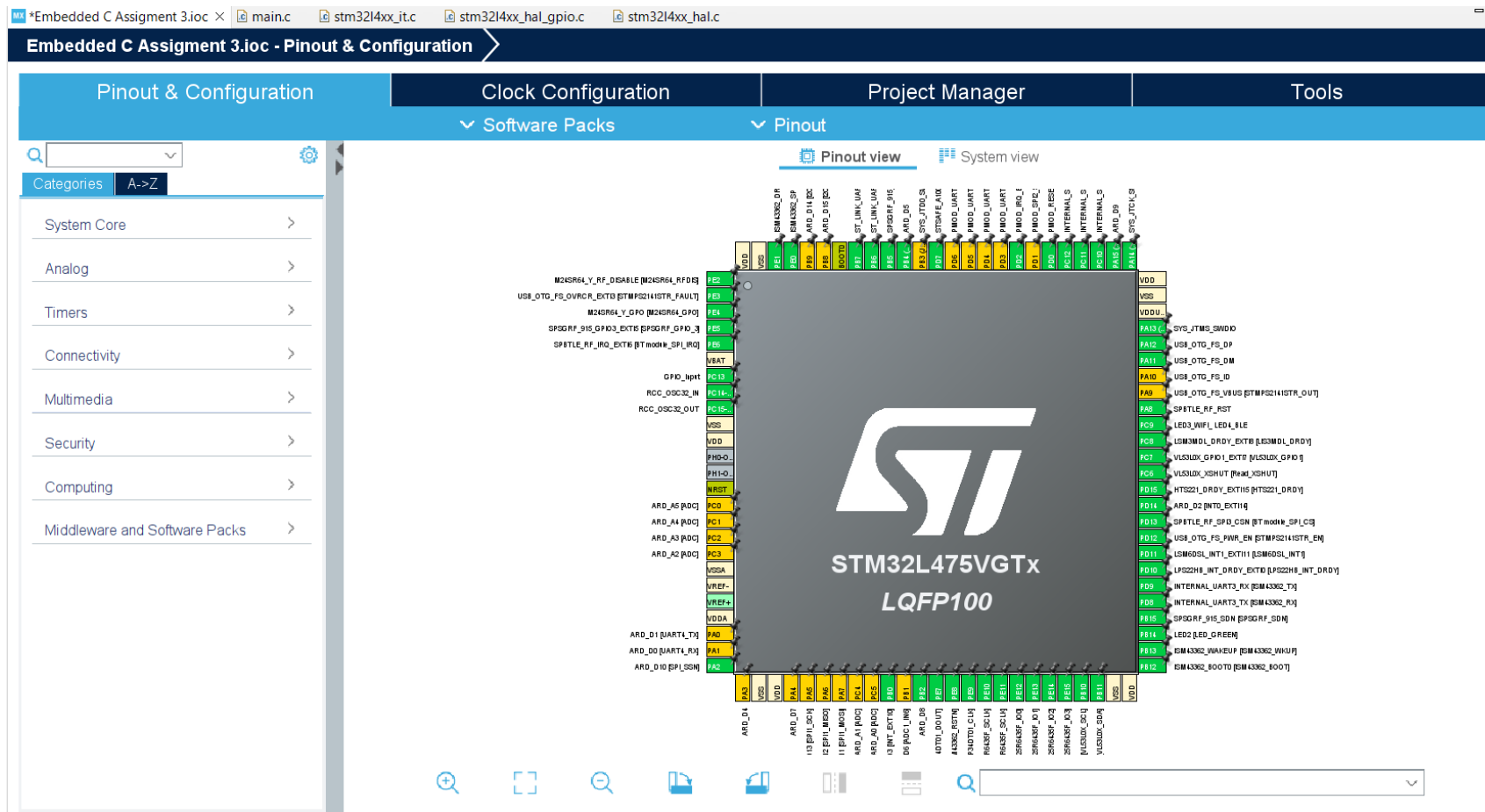
☒ Copy only the necessary library files

? < Back Next > Finish Cancel

Step 5. Click yes to initialize all peripherals to default



Step 6. When in .ioc file, click Pinout & Configurations

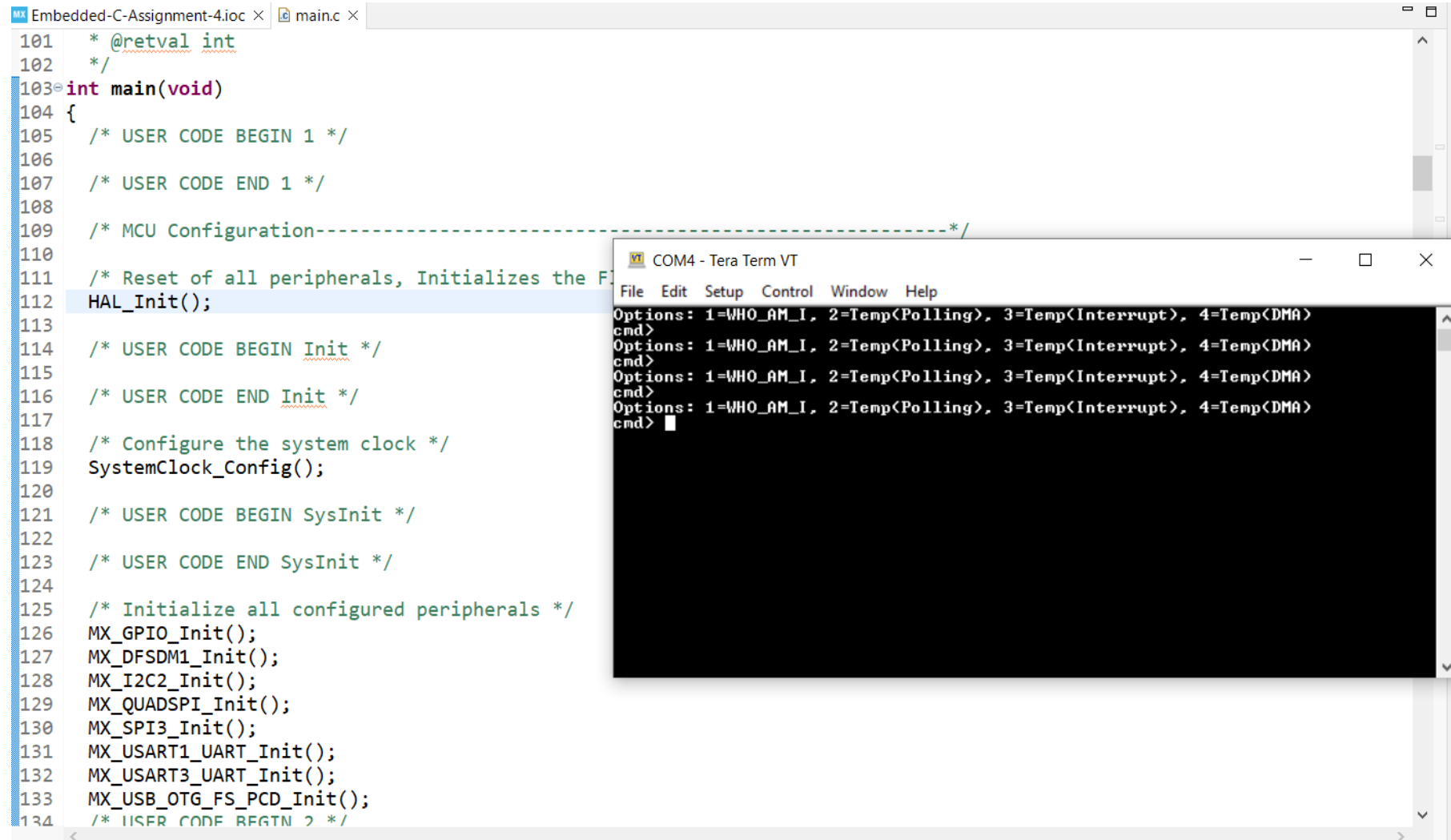


Step 8. User Story 1. CLI. Create a CLI (Command Line Interface) on UART1 that prompts you to enter a 1 for WHO_AM_I using polling, 2 temperature read using polling, and 3 for temperature readings using interrupts, and a 4 for temperature read using DMA>

```
Embedded-C-Assignment-4.ioc  main.c x
68 static void MX_USART1_UART_Init(void);
69 static void MX_USART3_UART_Init(void);
70 static void MX_USB_OTG_FS_PCD_Init(void);
71 /* USER CODE BEGIN PFP */
72
73 /* USER CODE END PFP */
74
75 /* Private user code -----
76 /* USER CODE BEGIN 0 */
77 void do_who_am_i()
78 {
79
80 }
81
82 void do_temp_polled()
83 {
84
85 }
86
87 void do_temp_interrupt()
88 {
89
90 }
91
92 void do_temp_dma()
93 {
94
95 }
96
97 /* USER CODE END 0 */
```

```
Embedded-C-Assignment-4.ioc  main.c x
138 /* Infinite loop */
139 /* USER CODE BEGIN WHILE */
140 while (1)
141 {
142     /* USER CODE END WHILE */
143
144     /* USER CODE BEGIN 3 */
145     // Issue command prompt
146     char *prompt = "Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)\n\r cmd> ";
147     HAL_UART_Transmit(&huart1, (uint8_t*)prompt, strlen(prompt), 1000);
148
149     // Wait for a single number entry
150     char ch;
151     HAL_UART_Receive(&huart1, (uint8_t*)&ch, 1, HAL_MAX_DELAY);
152
153     char *msg = "\r\n";
154     HAL_UART_Transmit(&huart1, (uint8_t*)msg, strlen(msg), 1000);
155
156     switch(ch)
157     {
158     case '1': /*msg= "\r\nTODO: WHO_AM_I\r\n";*/ do_who_am_i(); break;
159     case '2': /*msg= "\r\nTODO: Temp(Polling)\r\n";*/ do_temp_polled(); break;
160     case '3': /*msg= "\r\nTODO: Temp(Interrupt)\r\n";*/ do_temp_interrupt(); break;
161     case '4': /*msg= "\r\nTODO: Temp(DMA)\r\n";*/ do_temp_dma(); break;
162     // Fall through if none
163     }
164     // HAL_UART_Transmit(&huart1, (uint8_t*)msg, strlen(msg), 1000);
165
166 }
```

Step 9. User Story 1: Build and run in debug mode. Open tera term and test is successful



The screenshot shows an IDE with two windows. The main window displays a C program for an embedded system. The code includes comments for user code sections and initialization functions. The terminal window, titled 'COM4 - Tera Term VT', shows the output of the program, which is a repeated message: 'Options: 1=WHO_AM_I, 2=Temp<Polling>, 3=Temp<Interrupt>, 4=Temp<DMA>'.

```
101  * @retval int
102  */
103  int main(void)
104  {
105      /* USER CODE BEGIN 1 */
106
107      /* USER CODE END 1 */
108
109      /* MCU Configuration-----*/
110
111      /* Reset of all peripherals, Initializes the F
112      HAL_Init();
113
114      /* USER CODE BEGIN Init */
115
116      /* USER CODE END Init */
117
118      /* Configure the system clock */
119      SystemClock_Config();
120
121      /* USER CODE BEGIN SysInit */
122
123      /* USER CODE END SysInit */
124
125      /* Initialize all configured peripherals */
126      MX_GPIO_Init();
127      MX_DFSDM1_Init();
128      MX_I2C2_Init();
129      MX_QUADSPI_Init();
130      MX_SPI3_Init();
131      MX_USART1_UART_Init();
132      MX_USART3_UART_Init();
133      MX_USB_OTG_FS_PCD_Init();
134      /* USER CODE BEGIN 2 */
```

COM4 - Tera Term VT

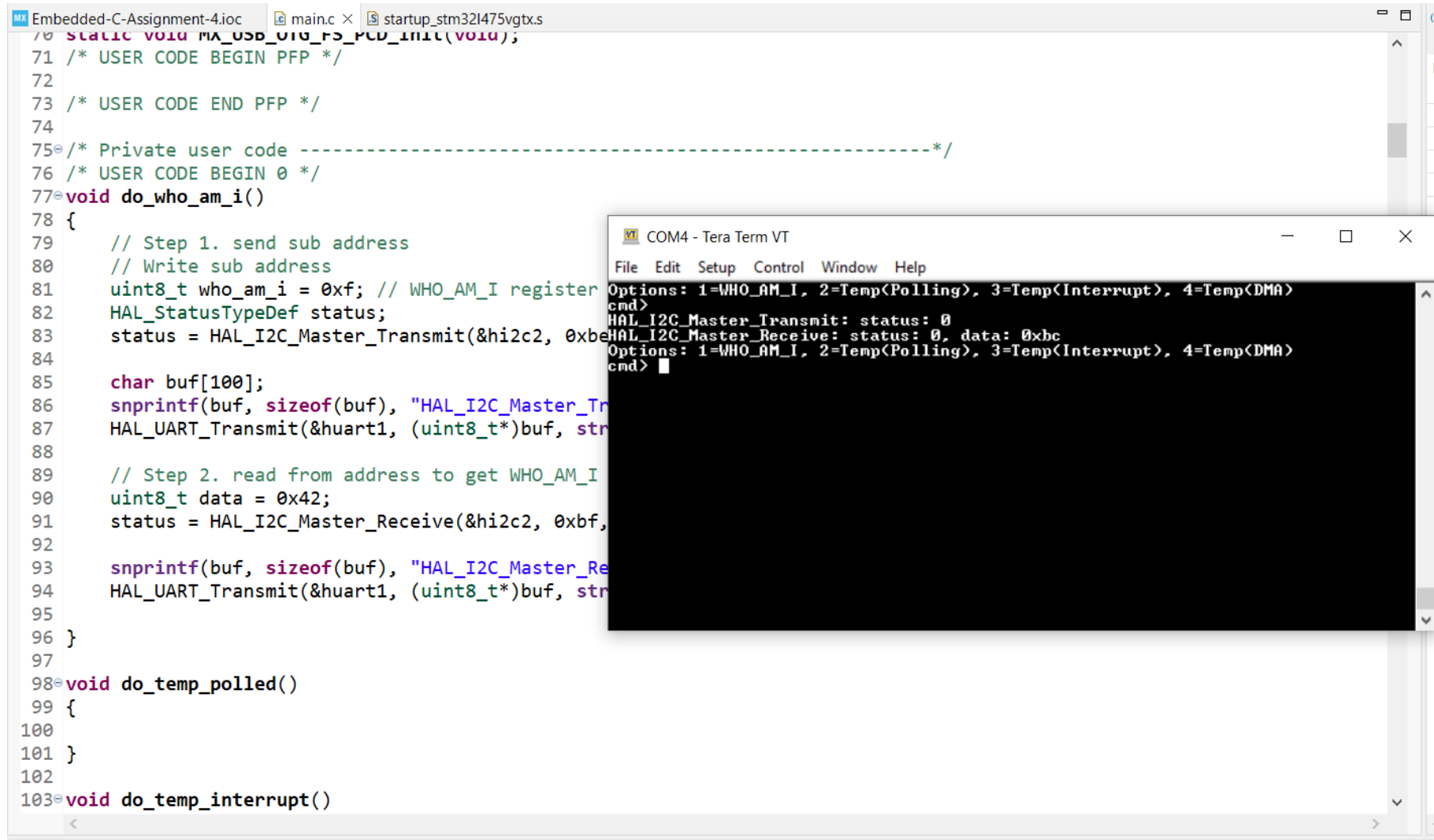
File Edit Setup Control Window Help

Options: 1=WHO_AM_I, 2=Temp<Polling>, 3=Temp<Interrupt>, 4=Temp<DMA>
cmd>
Options: 1=WHO_AM_I, 2=Temp<Polling>, 3=Temp<Interrupt>, 4=Temp<DMA>
cmd>
Options: 1=WHO_AM_I, 2=Temp<Polling>, 3=Temp<Interrupt>, 4=Temp<DMA>
cmd>
Options: 1=WHO_AM_I, 2=Temp<Polling>, 3=Temp<Interrupt>, 4=Temp<DMA>
cmd>

Step 10. User Story 2. WHO_AM_I. Polling. When the user selects 1, display contents of WHO_AM_I register from HTS221

```
Embedded-C-Assignment-4.ioc  main.c × startup_stm32l475vgtx.s
70 static void MX_USB_OTG_FS_PCD_Init(void);
71 /* USER CODE BEGIN PFP */
72
73 /* USER CODE END PFP */
74
75 /* Private user code -----*/
76 /* USER CODE BEGIN 0 */
77 void do_who_am_i()
78 {
79     // Step 1. send sub address
80     // Write sub address
81     uint8_t who_am_i = 0xf; // WHO_AM_I register
82     HAL_StatusTypeDef status;
83     status = HAL_I2C_Master_Transmit(&hi2c2, 0xbe, &who_am_i, sizeof(who_am_i), 1000);
84
85     char buf[100];
86     snprintf(buf, sizeof(buf), "HAL_I2C_Master_Transmit: status: %u\r\n", status);
87     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
88
89     // Step 2. read from address to get WHO_AM_I
90     uint8_t data = 0x42;
91     status = HAL_I2C_Master_Receive(&hi2c2, 0xbf, &data, sizeof(data), 1000);
92
93     snprintf(buf, sizeof(buf), "HAL_I2C_Master_Receive: status: %u, data: 0x%x\r\n", status, data);
94     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
95
96 }
```

Step 11. User Story 2: Build and run the code, test is successful



The image shows a screenshot of a development environment. On the left, a code editor window titled 'Embedded-C-Assignment-4.ioc' displays C code. The code includes comments for 'USER CODE BEGIN PFP' and 'Private user code'. The function `do_who_am_i()` is defined, which sends a sub address via I2C, reads data from the WHO_AM_I register, and prints the status and data. The function `do_temp_polled()` and `do_temp_interrupt()` are also declared. On the right, a terminal window titled 'COM4 - Tera Term VT' shows the output of the program. The terminal displays the status of I2C Master Transmit and Receive operations, indicating successful transmission and reception of data.

```
70 static void MX_USB_OTG_FS_PCD_Init(void);
71 /* USER CODE BEGIN PFP */
72
73 /* USER CODE END PFP */
74
75 /* Private user code -----*/
76 /* USER CODE BEGIN 0 */
77 void do_who_am_i()
78 {
79     // Step 1. send sub address
80     // Write sub address
81     uint8_t who_am_i = 0xf; // WHO_AM_I register
82     HAL_StatusTypeDef status;
83     status = HAL_I2C_Master_Transmit(&hi2c2, 0x42, 0xf, 1, 1000);
84
85     char buf[100];
86     snprintf(buf, sizeof(buf), "HAL_I2C_Master_Transmit: status: %d", status);
87     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
88
89     // Step 2. read from address to get WHO_AM_I
90     uint8_t data = 0x42;
91     status = HAL_I2C_Master_Receive(&hi2c2, 0x42, &data, 1, 1000);
92
93     snprintf(buf, sizeof(buf), "HAL_I2C_Master_Receive: status: %d, data: %x", status, data);
94     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
95 }
96
97 void do_temp_polled()
98 {
99 }
100
101 void do_temp_interrupt()
```

```
COM4 - Tera Term VT
File Edit Setup Control Window Help
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
HAL_I2C_Master_Transmit: status: 0
HAL_I2C_Master_Receive: status: 0, data: 0xf
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
```

Step 12. User Story 3. Temperature. Polling. The user selects 2, use polling to read HTS221 and display temperature on console.

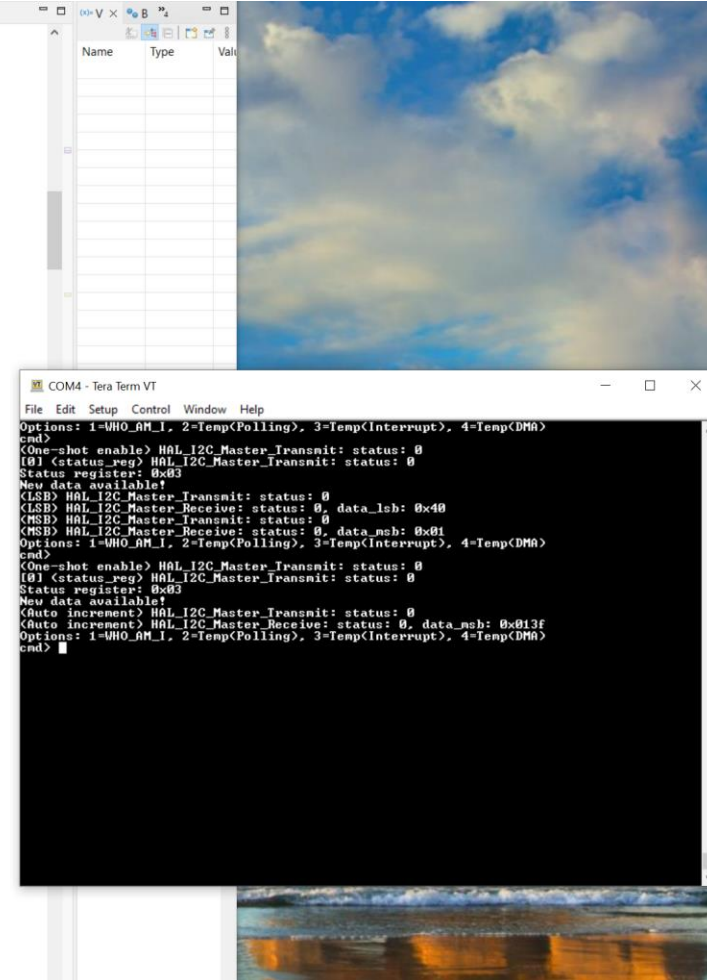
```
*main.c ×
70 static void MX_USB_OTG_FS_PCD_Init(void);
71 /* USER CODE BEGIN PFP */
72
73 /* USER CODE END PFP */
74
75 /* Private user code -----*/
76 /* USER CODE BEGIN 0 */
77 #define HST221_READ_ADDRESS 0xbf
78 #define HST221_WRITE_ADDRESS 0xbe
79
80 void do_who_am_i()
100
101 void do_temp_polled()
102 {
103     // Setup control register 1
104     uint8_t control_reg1 = 0x20;
105     uint8_t control_data1[] = {control_reg1, 0x85}; // output registers not updated until MSB and LSB reading, 1 Hz
106     HAL_StatusTypeDef status;
107     status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, control_data1, sizeof(control_data1), 1000);
108
109     // Start a conversion
110     uint8_t control_reg2 = 0x21;
111     uint8_t control_data2[] = {control_reg2, 0x01};
112     status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, control_data2, sizeof(control_data2), 1000);
113
114     char buf[100];
115     snprintf(buf, sizeof(buf), "(One-shot enable) HAL_I2C_Master_Transmit: status: %u\r\n", status);
116     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
117
118     // Wait for conversion complete
119     uint8_t status_reg = 0x27;
120     uint8_t status_data = 0;
121     int count = 0;
122     while(count < 10)
123     {
124         // Send read status register sub command
125         status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &status_reg, sizeof(status_reg), 1000);
126         snprintf(buf, sizeof(buf), "[%d] (status_reg) HAL_I2C_Master_Transmit: status: %u\r\n", count, status);
127         HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
128
129         // Read conversion status
130         status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&status_data, sizeof(status_data), 1000);
131         snprintf(buf, sizeof(buf), "Status register: 0x%02x\r\n", status_data);
132         HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
133
134         // Check for temperature conversion complete
135         if(status_data & 0x1)
136         {
137             snprintf(buf, sizeof(buf), "New data available!\r\n");
138             HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
139             break;
140         }
141         HAL_Delay(1000);
142         count++;
143     }
144 }
```


Step 13. User Story 3: Toggle between auto increment and direct read from LSB and MSB

```
@ *main.c x
137     snprintf(buf, sizeof(buf), "New data available!\r\n");
138     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
139     break;
140 }
141 HAL_Delay(1000);
142 count++;
143 }
144
145 // Toggle between normal poll and address increment poll
146 static int toggle = 1;
147
148 if(toggle)
149 {
150     toggle = 0;
151
152     // Read temperature LSB
153     uint8_t temperature_lsb = 0x2a;
154     status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &temperature_lsb, sizeof(temperature_lsb), 1000);
155     snprintf(buf, sizeof(buf), "(LSB) HAL_I2C_Master_Transmit: status: %u\r\n", status);
156     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
157
158     uint8_t data_lsb = 0x42;
159     status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data_lsb, sizeof(data_lsb), 1000);
160     snprintf(buf, sizeof(buf), "(LSB) HAL_I2C_Master_Receive: status: %u, data_lsb: 0x%02x\r\n", status, data_lsb);
161     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
162
163     // Read temperature MSB
164     uint8_t temperature_msb = 0x2b;
165     status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &temperature_msb, sizeof(temperature_msb), 1000);
166     snprintf(buf, sizeof(buf), "(MSB) HAL_I2C_Master_Transmit: status: %u\r\n", status);
167     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
168
169     uint8_t data_msb = 0x42;
170     status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data_msb, sizeof(data_msb), 1000);
171     snprintf(buf, sizeof(buf), "(MSB) HAL_I2C_Master_Receive: status: %u, data_msb: 0x%02x\r\n", status, data_msb);
172     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
173 }
174 else
175 {
176     toggle = 1;
177     // Read using auto increment
178     uint8_t temperature_lsb = 0x2a | 0x80;
179     status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &temperature_lsb, sizeof(temperature_lsb), 1000);
180     snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Transmit: status: %u\r\n", status);
181     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
182
183     uint16_t data = 0x4242;
184     status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data, sizeof(data), 1000);
185     snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Receive: status: %u, data_msb: 0x%04x\r\n", status, data);
186     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
187 }
188
189 }
```

Step 14. User Story 3: Build and run the code, test is successful

```
@ 'main.c' x
137     snprintf(buf, sizeof(buf), "New data available!\r\n");
138     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
139     break;
140 }
141 HAL_Delay(1000);
142 count++;
143 }
144
145 // Toggle between normal poll and address increment poll
146 static int toggle = 1;
147
148 if(toggle)
149 {
150     toggle = 0;
151
152     // Read temperature LSB
153     uint8_t temperature_lsb = 0x2a;
154     status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &temperature_lsb, sizeof(temperature_lsb), 1000);
155     snprintf(buf, sizeof(buf), "(LSB) HAL_I2C_Master_Transmit: status: %u\r\n", status);
156     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
157
158     uint8_t data_lsb = 0x42;
159     status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data_lsb, sizeof(data_lsb), 1000);
160     snprintf(buf, sizeof(buf), "(LSB) HAL_I2C_Master_Receive: status: %u, data_lsb: 0x%02x\r\n", status, data_lsb);
161     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
162
163     // Read temperature MSB
164     uint8_t temperature_msb = 0x2b;
165     status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &temperature_msb, sizeof(temperature_msb), 1000);
166     snprintf(buf, sizeof(buf), "(MSB) HAL_I2C_Master_Transmit: status: %u\r\n", status);
167     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
168
169     uint8_t data_msb = 0x42;
170     status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data_msb, sizeof(data_msb), 1000);
171     snprintf(buf, sizeof(buf), "(MSB) HAL_I2C_Master_Receive: status: %u, data_msb: 0x%02x\r\n", status, data_msb);
172     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
173 }
174 else
175 {
176     toggle = 1;
177     // Read using auto increment
178     uint8_t temperature_lsb = 0x2a | 0x80;
179     status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &temperature_lsb, sizeof(temperature_lsb), 1000);
180     snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Transmit: status: %u\r\n", status);
181     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
182
183     uint16_t data = 0x4242;
184     status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data, sizeof(data), 1000);
185     snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Receive: status: %u, data_msb: 0x%04x\r\n", status, data);
186     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
187 }
188
189 }
190 }
```



Step 15. User Story 4. Temperature. Interrupts. When the user selects 3, use interrupts to read from HTS221 and display temperature on console.

```
main.c x
76 / USER CODE BEGIN 0 /
77 #define HST221_READ_ADDRESS 0xbf
78 #define HST221_WRITE_ADDRESS 0xbe
79
80 static uint8_t irq_complete = 0;
81 static uint8_t status_flag = 0;
82 static uint16_t data = 0;
83 static uint8_t status_data = 0;
84
85 void do_who_am_i()
105
106 void do_temp_polled()
195
196 void do_temp_interrupt()
197 {
198     irq_complete = 0;
199     char buf[100];
200     // Setup control register 1
201     uint8_t control_reg1 = 0x20;
202     uint8_t control_data1[] = {control_reg1, 0x85}; // output registers not updated until MSB and LSB reading, 1 Hz
203     HAL_StatusTypeDef status;
204     status = HAL_I2C_Master_Transmit_IT(&hi2c2, HST221_WRITE_ADDRESS, control_data1, sizeof(control_data1));
205     snprintf(buf, sizeof(buf), "(Control register 1) HAL_I2C_Master_Transmit_IT: status: %u\r\n", status);
206     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
207     while(0 == irq_complete)
208     {
209         HAL_Delay(1000);
210     }
211
212     irq_complete = 0;
213     // Start a conversion but interrupt driven
214     int control_reg2 = 0x21;
215     uint8_t control_data2[] = {control_reg2, 0x01}; // One-shot enable
216     status = HAL_I2C_Master_Transmit_IT(&hi2c2, HST221_WRITE_ADDRESS, control_data2, sizeof(control_data2));
```

```
main.c x
217
218     irq_complete = 0;
219     // Start a conversion but interrupt driven
220     int control_reg2 = 0x21;
221     uint8_t control_data2[] = {control_reg2, 0x01}; // One-shot enable
222     status = HAL_I2C_Master_Transmit_IT(&hi2c2, HST221_WRITE_ADDRESS, control_data2, sizeof(control_data2));
223     snprintf(buf, sizeof(buf), "(One-shot Enable) HAL_I2C_Master_Transmit_IT: status: %u\r\n", status);
224     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
225     while(0 == irq_complete)
226     {
227         HAL_Delay(1000);
228     }
229
230     irq_complete = 0;
231     // Wait for conversion complete
232     uint8_t status_reg = 0x27;
233     int count = 0;
234     while(count < 10)
235     {
236         // Send read status register sub command
237         status = HAL_I2C_Master_Transmit_IT(&hi2c2, HST221_WRITE_ADDRESS, &status_reg, sizeof(status_reg));
238         snprintf(buf, sizeof(buf), "[%d] (status_reg) HAL_I2C_Master_Transmit_IT: status: %u\r\n", count, status);
239         HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
240         while(0 == irq_complete)
241         {
242             HAL_Delay(1000);
243         }
244
245         irq_complete = 0;
246         status_flag = 1;
247         // Read conversion status
248         status = HAL_I2C_Master_Receive_IT(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&status_data, sizeof(status_data));
249         while(0 == irq_complete)
250         {
251             HAL_Delay(1000);
252         }
253
254         irq_complete = 0;
255         status_flag = 0;
256         // Read conversion data
257         status = HAL_I2C_Master_Receive_IT(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data, sizeof(data));
258         while(0 == irq_complete)
259         {
260             HAL_Delay(1000);
261         }
262
263         snprintf(buf, sizeof(buf), "Temperature: %d\r\n", data);
264         HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
265     }
266 }
```


Step 16. User Story 4: I code in the way that when we use Rx or Tx interrupt mode, we check if the irq is complete or not. If it is complete, we can run to the next line of code

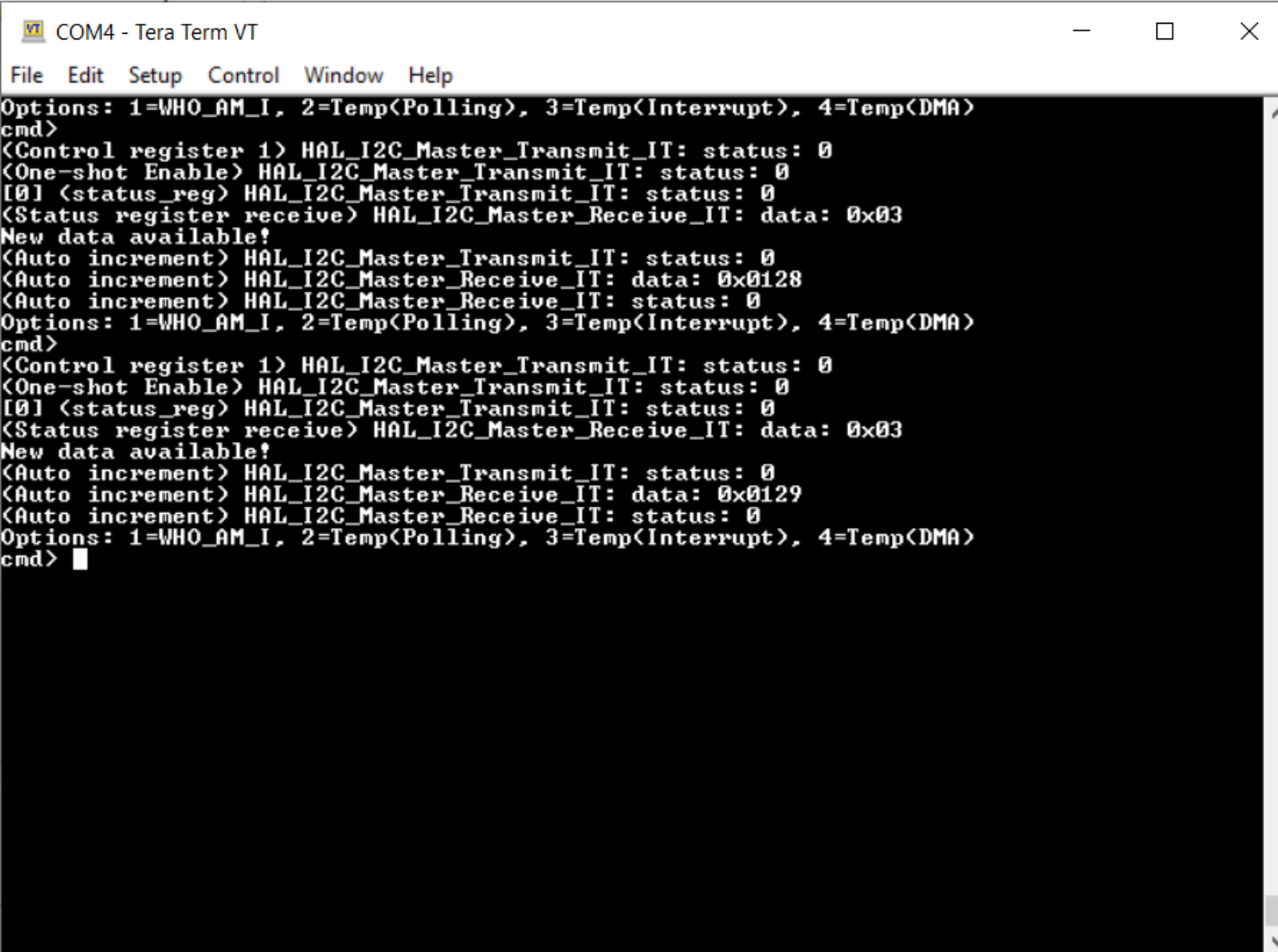
```
main.c X
225 // Wait for conversion complete
226 uint8_t status_reg = 0x27;
227 int count = 0;
228 while(count < 10)
229 {
230     // Send read status register sub command
231     status = HAL_I2C_Master_Transmit_IT(&hi2c2, HST221_WRITE_ADDRESS, &status_reg, sizeof(status_reg));
232     snprintf(buf, sizeof(buf), "[%d] (status_reg) HAL_I2C_Master_Transmit_IT: status: %u\r\n", count, status);
233     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
234     while(0 == irq_complete)
235     {
236         HAL_Delay(1000);
237     }
238
239     irq_complete = 0;
240     status_flag = 1;
241     // Read conversion status
242     status = HAL_I2C_Master_Receive_IT(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&status_data, sizeof(status_data));
243     while(0 == irq_complete)
244     {
245         HAL_Delay(1000);
246     }
247
248     // Check for temperature conversion complete
249     if(status_data & 0x1)
250     {
251         snprintf(buf, sizeof(buf), "New data available!\r\n");
252         HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
253         break;
254     }
255     HAL_Delay(1000);
256     count++;
257 }
258
```

```
main.c X
252 HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
253     break;
254 }
255 HAL_Delay(1000);
256 count++;
257 }
258
259
260 irq_complete = 0;
261 // Read using auto increment
262 uint8_t temperature_lsb = 0x2a | 0x80;
263 status = HAL_I2C_Master_Transmit_IT(&hi2c2, HST221_WRITE_ADDRESS, &temperature_lsb, sizeof(temperature_lsb));
264 snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Transmit_IT: status: %u\r\n", status);
265 HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
266 while(0 == irq_complete)
267 {
268     HAL_Delay(1000);
269 }
270
271 irq_complete = 0;
272 // Receive using interrupt
273 status = HAL_I2C_Master_Receive_IT(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data, sizeof(data));
274 snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Receive_IT: status: %u\r\n", status);
275 HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
276 while(0 == irq_complete)
277 {
278     HAL_Delay(1000);
279 }
280 }
281
```

Step 17. User Story 4: Tx and Rx call back function implementation

```
main.c x
871 /* USER CODE BEGIN MX_GPIO_Init_2 */
872 /* USER CODE END MX_GPIO_Init_2 */
873 }
874
875 /* USER CODE BEGIN 4 */
876 void HAL_I2C_MasterTxCpltCallback(I2C_HandleTypeDef *hi2c)
877 {
878     HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
879
880     irq_complete = 1;
881 }
882
883 void HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef *hi2c)
884 {
885     char buf[100];
886     if(status_flag == 1)
887     {
888         status_flag = 0;
889         snprintf(buf, sizeof(buf), "(Status register receive) HAL_I2C_Master_Receive_IT: data: 0x%02x\r\n", status_data);
890         HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
891     }
892     else
893     {
894         snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Receive_IT: data: 0x%04x\r\n", data);
895         HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
896     }
897     irq_complete = 1;
898 }
```

Step 18. User Story 4: Build and run the code, test is successful. I use auto increment to read the temperature data

A screenshot of a Tera Term VT window titled "COM4 - Tera Term VT". The window has a menu bar with "File", "Edit", "Setup", "Control", "Window", and "Help". The main area displays a log of I2C transactions. The log shows two identical sequences of events. Each sequence starts with a command prompt "cmd>" followed by a series of status and data reports: "<Control register 1> HAL_I2C_Master_Transmit_IT: status: 0", "<One-shot Enable> HAL_I2C_Master_Transmit_IT: status: 0", "[0] <status_reg> HAL_I2C_Master_Transmit_IT: status: 0", "<Status register receive> HAL_I2C_Master_Receive_IT: data: 0x03", "New data available!", "<Auto increment> HAL_I2C_Master_Transmit_IT: status: 0", "<Auto increment> HAL_I2C_Master_Receive_IT: data: 0x0128", "<Auto increment> HAL_I2C_Master_Receive_IT: status: 0", and "Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)". The log ends with another "cmd>" prompt. The window has standard Windows window controls (minimize, maximize, close) in the top right corner and a vertical scrollbar on the right side.

```
COM4 - Tera Term VT
File Edit Setup Control Window Help
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
<Control register 1> HAL_I2C_Master_Transmit_IT: status: 0
<One-shot Enable> HAL_I2C_Master_Transmit_IT: status: 0
[0] <status_reg> HAL_I2C_Master_Transmit_IT: status: 0
<Status register receive> HAL_I2C_Master_Receive_IT: data: 0x03
New data available!
<Auto increment> HAL_I2C_Master_Transmit_IT: status: 0
<Auto increment> HAL_I2C_Master_Receive_IT: data: 0x0128
<Auto increment> HAL_I2C_Master_Receive_IT: status: 0
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
<Control register 1> HAL_I2C_Master_Transmit_IT: status: 0
<One-shot Enable> HAL_I2C_Master_Transmit_IT: status: 0
[0] <status_reg> HAL_I2C_Master_Transmit_IT: status: 0
<Status register receive> HAL_I2C_Master_Receive_IT: data: 0x03
New data available!
<Auto increment> HAL_I2C_Master_Transmit_IT: status: 0
<Auto increment> HAL_I2C_Master_Receive_IT: data: 0x0129
<Auto increment> HAL_I2C_Master_Receive_IT: status: 0
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
```

Step 19. User Story 5. Temperature. DMA. When the user selects 4, use DMA to read from HTS221 and display temperature on console. Modify the DMA1 in .ioc file first

main.c *Embedded-C-Assignment-4.ioc

Embedded-C-Assignment-4.ioc - Pinout & Configuration

Categories A-Z

System Core

- DMA
- GPIO
- IWDG
- NVIC
- RCC
- SYS
- TSC
- WWDG

Analog

Timers

Connectivity

Multimedia

Security

Computing

Middleware ...

Pinout & Configuration

DMA Mode and Configuration

Configuration

DMA Request	Channel	Direction	Priority
I2C2_RX	DMA1 Channel 5	Peripheral To Memory	Low
I2C2_TX	DMA1 Channel 4	Memory To Peripheral	Low

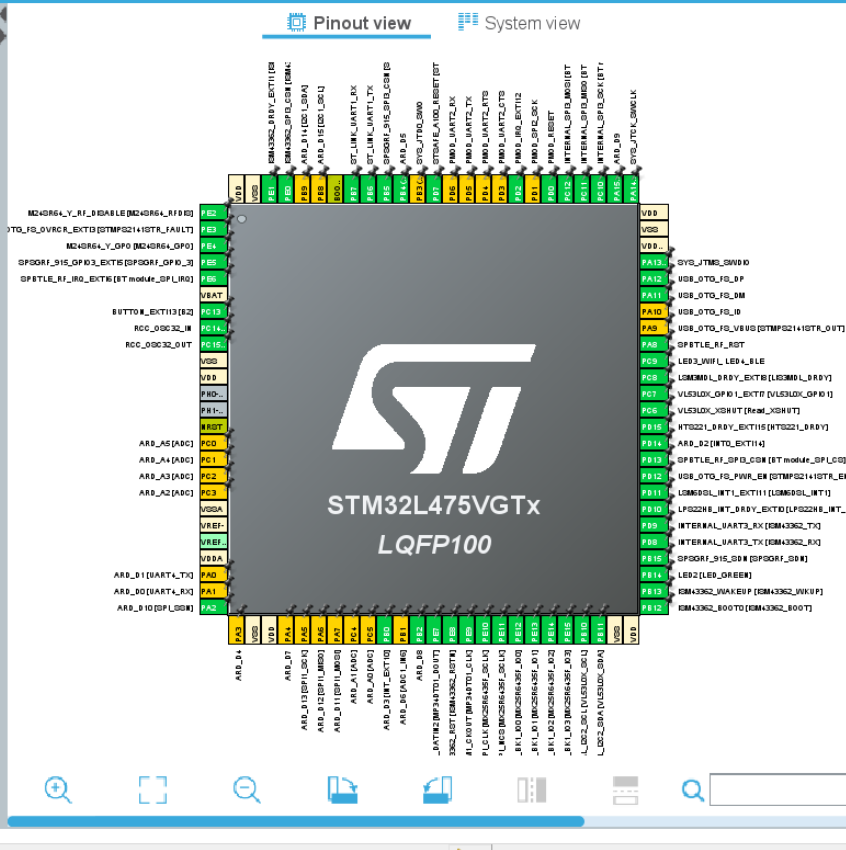
Add Delete

DMA Request Settings

Mode	Peripheral	Increment Address	Data Width
Normal		<input type="checkbox"/>	Byte

Pinout view

STM32L475VGTx LQFP100



File Explorer

Step 20. User Story 5. Write the code in main.c file, it is very similar to the interrupt version

```
main.c x stm32l4xx_hal_msp.c
199*void do_temp_interrupt()
284
285*void do_temp_dma()
286 {
287     irq_complete = 0;
288     char buf[100];
289     // Setup control register 1
290     uint8_t control_reg1 = 0x20;
291     uint8_t control_data1[] = {control_reg1, 0x85}; // output registers not updated until MSB and LSB reading, 1 Hz
292     HAL_StatusTypeDef status;
293     status = HAL_I2C_Master_Transmit_DMA(&hi2c2, HST221_WRITE_ADDRESS, control_data1, sizeof(control_data1));
294     snprintf(buf, sizeof(buf), "(Control register 1) HAL_I2C_Master_Transmit_DMA: status: %u\r\n", status);
295     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
296     while(0 == irq_complete)
297     {
298         HAL_Delay(1000);
299     }
300
301     irq_complete = 0;
302     // Start a conversion but interrupt driven
303     int control_reg2 = 0x21;
304     uint8_t control_data2[] = {control_reg2, 0x01}; // One-shot enable
305     status = HAL_I2C_Master_Transmit_DMA(&hi2c2, HST221_WRITE_ADDRESS, control_data2, sizeof(control_data2));
306     snprintf(buf, sizeof(buf), "(One-shot Enable) HAL_I2C_Master_Transmit_DMA: status: %u\r\n", status);
307     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
308     while(0 == irq_complete)
309     {
310         HAL_Delay(1000);
311     }
312     irq_complete = 0;
313 }
```

Step 21. User Story 5. Write the code in main.c file, continue

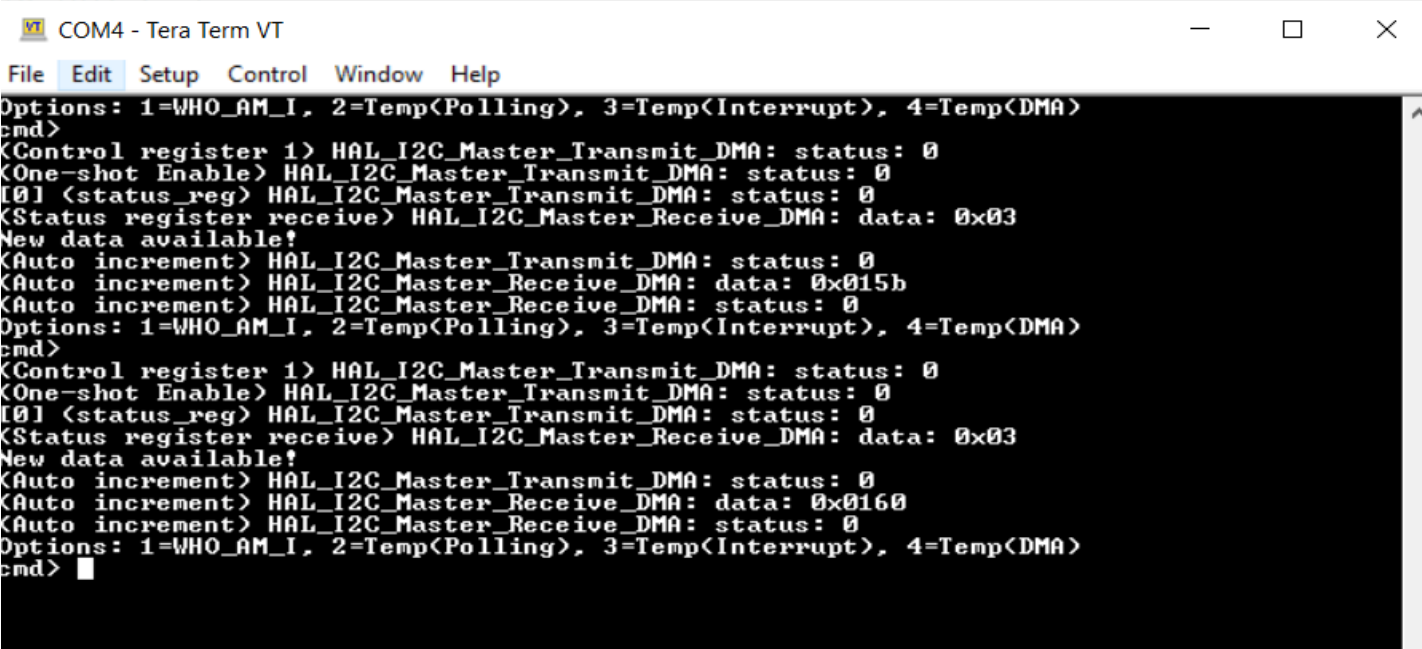
```
main.c x stm32l4xx_hal_msp.c
310     HAL_Delay(1000);
311 }
312 irq_complete = 0;
313 // Wait for conversion complete
314 uint8_t status_reg = 0x27;
315 int count = 0;
316 while(count < 10)
317 {
318     // Send read status register sub command
319     status = HAL_I2C_Master_Transmit_DMA(&hi2c2, HST221_WRITE_ADDRESS, &status_reg, sizeof(status_reg));
320     snprintf(buf, sizeof(buf), "[%d] (status_reg) HAL_I2C_Master_Transmit_DMA: status: %u\r\n", count, status);
321     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
322     while(0 == irq_complete)
323     {
324         HAL_Delay(1000);
325     }
326
327     irq_complete = 0;
328     status_flag = 1;
329     // Read conversion status
330     status = HAL_I2C_Master_Receive_DMA(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&status_data, sizeof(status_data));
331     while(0 == irq_complete)
332     {
333         HAL_Delay(1000);
334     }
335
336     // Check for temperature conversion complete
337     if(status_data & 0x1)
338     {
339         snprintf(buf, sizeof(buf), "New data available!\r\n");
340         HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
341         break;
342     }
343     HAL_Delay(1000);
}
```

Step 22. User Story 5. Write the code in main.c file, continue

```
main.c x stm32l4xx_hal_msp.c
343     HAL_Delay(1000);
344     count++;
345 }
346 irq_complete = 0;
347 // Read using auto increment
348 uint8_t temperature_lsb = 0x2a | 0x80;
349 status = HAL_I2C_Master_Transmit_DMA(&hi2c2, HST221_WRITE_ADDRESS, &temperature_lsb, sizeof(temperature_lsb));
350 snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Transmit_DMA: status: %u\r\n", status);
351 HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
352 while(0 == irq_complete)
353 {
354     HAL_Delay(1000);
355 }
356 irq_complete = 0;
357 // Receive using interrupt
358 status = HAL_I2C_Master_Receive_DMA(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data, sizeof(data));
359 snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Receive_DMA: status: %u\r\n", status);
360 HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
361 while(0 == irq_complete)
362 {
363     HAL_Delay(1000);
364 }
365 }
```

```
...
975 /* USER CODE BEGIN 4 */
976 void HAL_I2C_MasterTxCpltCallback(I2C_HandleTypeDef *hi2c)
977 {
978     HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
979
980     irq_complete = 1;
981 }
982
983 void HAL_I2C_MasterRxTxCpltCallback(I2C_HandleTypeDef *hi2c)
984 {
985     char buf[100];
986     if(status_flag == 1)
987     {
988         status_flag = 0;
989         snprintf(buf, sizeof(buf), "(Status register receive) HAL_I2C_Master_Receive_DMA: data: 0x%02x\r\n", status_data);
990         HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
991     }
992     else
993     {
994         snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Receive_DMA: data: 0x%04x\r\n", data);
995         HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
996     }
997     irq_complete = 1;
998 }
999 /* USER CODE END 4 */
```

Step 23. User Story 5. Build and run the code, test is successful



```
COM4 - Tera Term VT
File Edit Setup Control Window Help
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
<Control register 1> HAL_I2C_Master_Transmit_DMA: status: 0
<One-shot Enable> HAL_I2C_Master_Transmit_DMA: status: 0
[0] <status_reg> HAL_I2C_Master_Transmit_DMA: status: 0
<Status register receive> HAL_I2C_Master_Receive_DMA: data: 0x03
New data available!
<Auto increment> HAL_I2C_Master_Transmit_DMA: status: 0
<Auto increment> HAL_I2C_Master_Receive_DMA: data: 0x015b
<Auto increment> HAL_I2C_Master_Receive_DMA: status: 0
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd>
<Control register 1> HAL_I2C_Master_Transmit_DMA: status: 0
<One-shot Enable> HAL_I2C_Master_Transmit_DMA: status: 0
[0] <status_reg> HAL_I2C_Master_Transmit_DMA: status: 0
<Status register receive> HAL_I2C_Master_Receive_DMA: data: 0x03
New data available!
<Auto increment> HAL_I2C_Master_Transmit_DMA: status: 0
<Auto increment> HAL_I2C_Master_Receive_DMA: data: 0x0160
<Auto increment> HAL_I2C_Master_Receive_DMA: status: 0
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA)
cmd> █
```


Appendix, schematic for HTS221

