

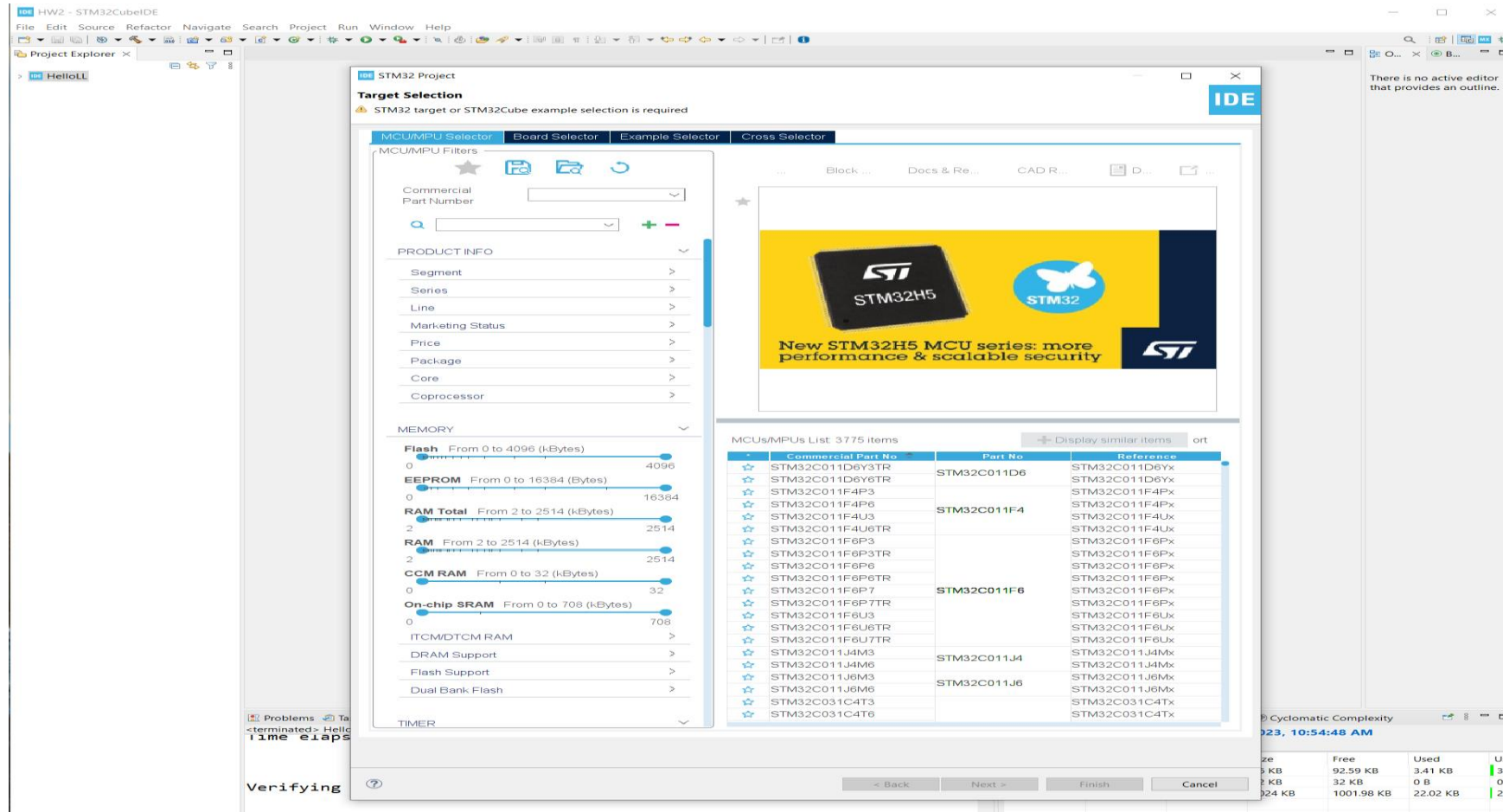
# UCSD Embedded C Assignment 2

By

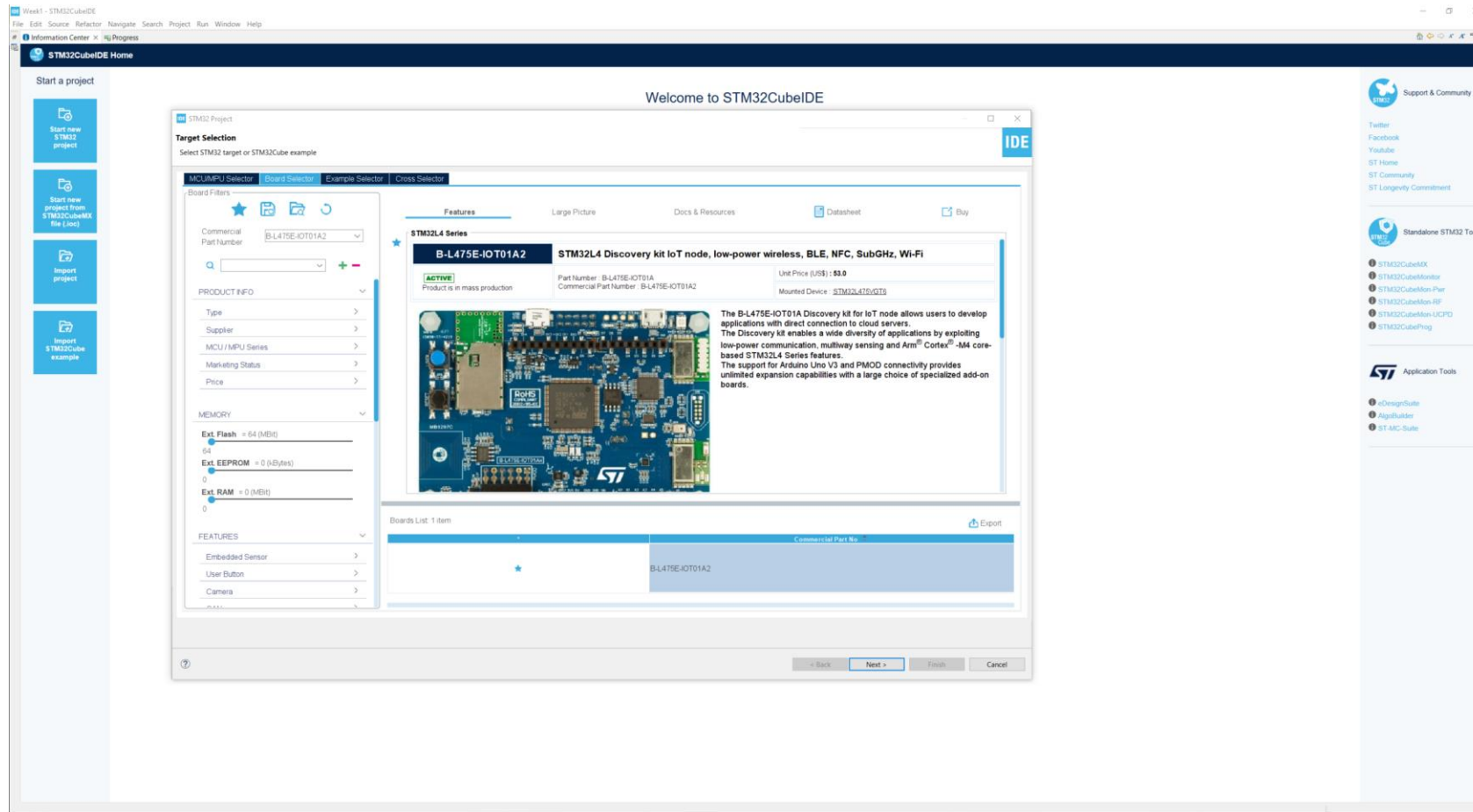
Hsuankai Chang

[hsuankac@umich.edu](mailto:hsuankac@umich.edu)

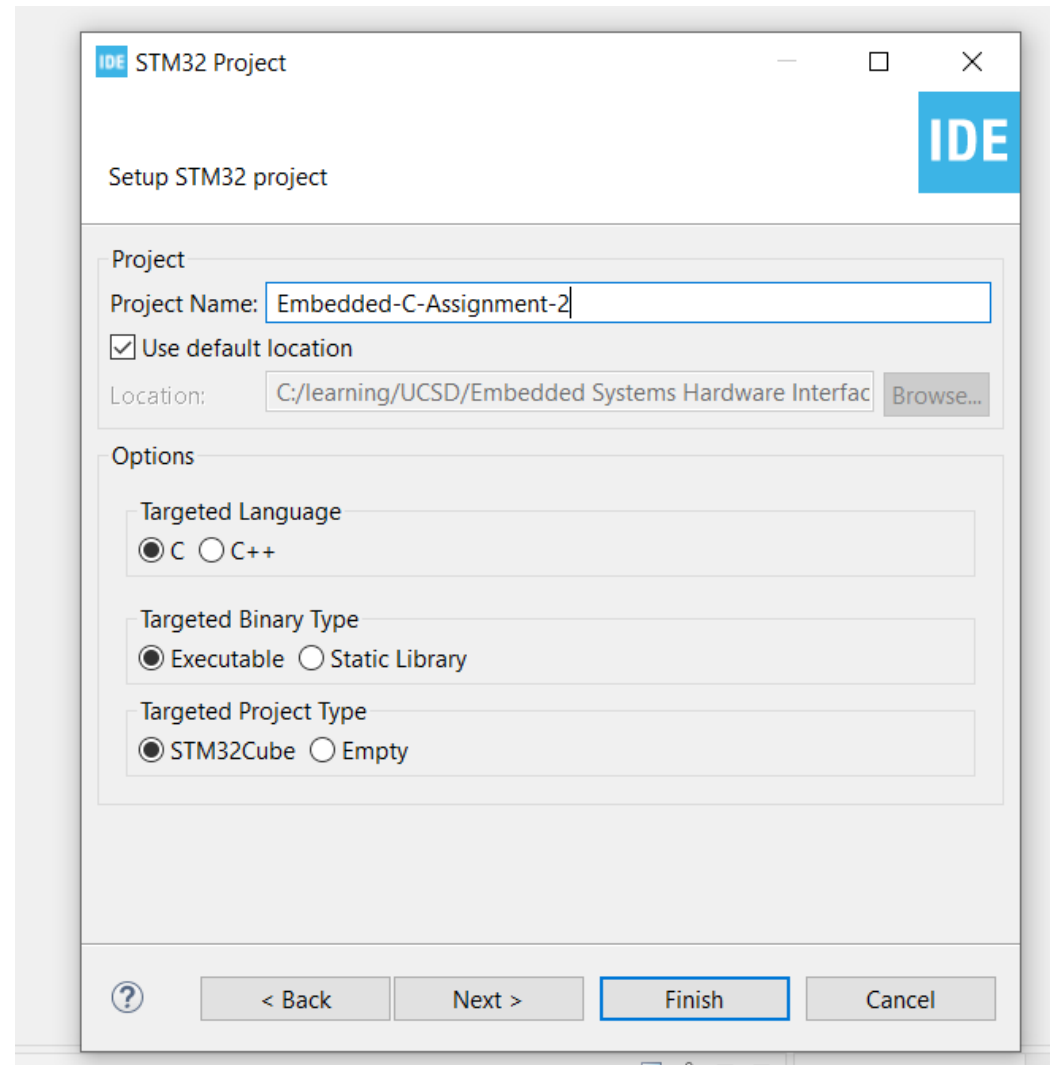
# Step 1. Startup STM32CubeIDE and create new STM32 project



# 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 an IDE. The window title is 'STM32 Project'. The main heading is 'Setup STM32 project'. The 'Project' section contains a 'Project Name' field with the text 'Embedded-C-Assignment-2', a checked 'Use default location' checkbox, and a 'Location' field with the path 'C:/learning/UCSD/Embedded Systems Hardware Interfac' and a 'Browse...' button. The 'Options' section contains three groups of radio buttons: 'Targeted Language' with 'C' selected, 'Targeted Binary Type' with 'Executable' selected, and 'Targeted Project Type' with 'STM32Cube' selected. At the bottom, there are buttons for '?', '< Back', 'Next >', 'Finish' (which is highlighted with a blue border), and 'Cancel'.

STM32 Project

Setup STM32 project

Project

Project Name: Embedded-C-Assignment-2

☒ Use default location

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

Options

Targeted Language

☒ C ☐ C++

Targeted Binary Type

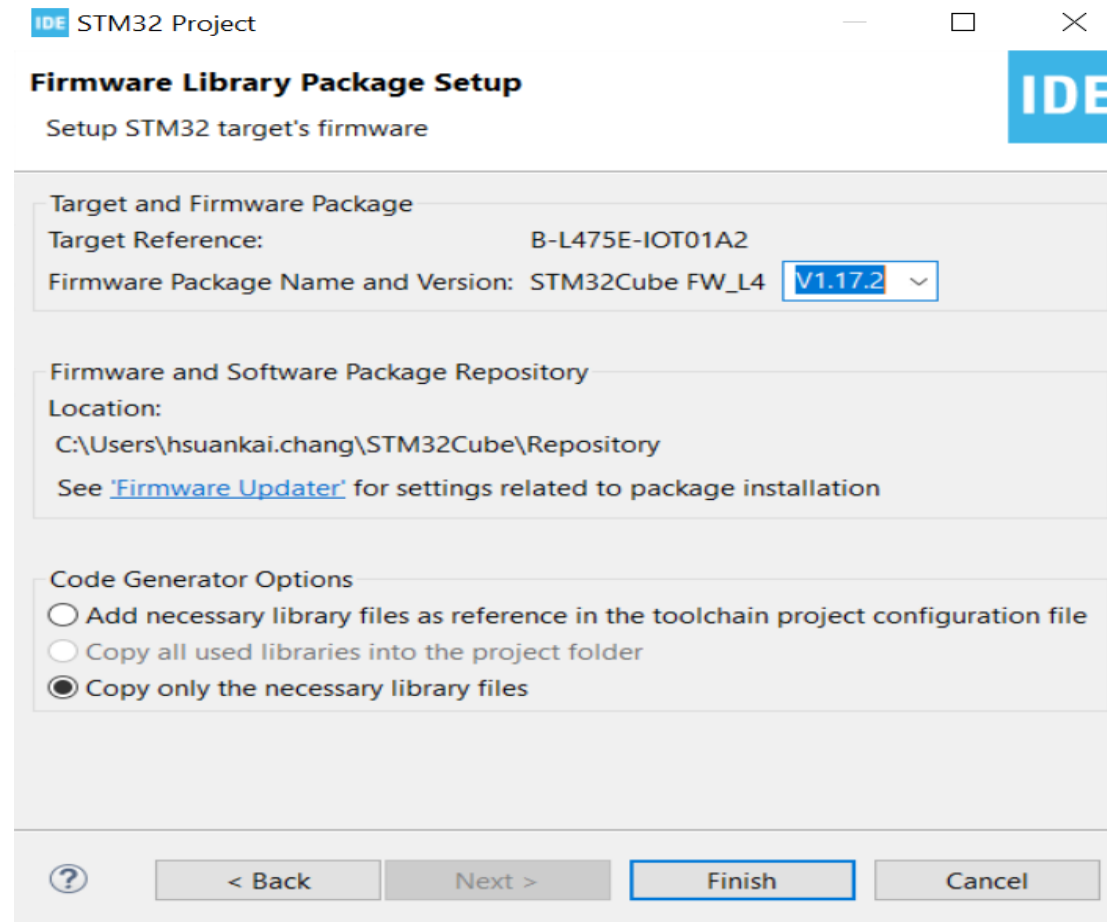
☒ Executable ☐ Static Library

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" (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

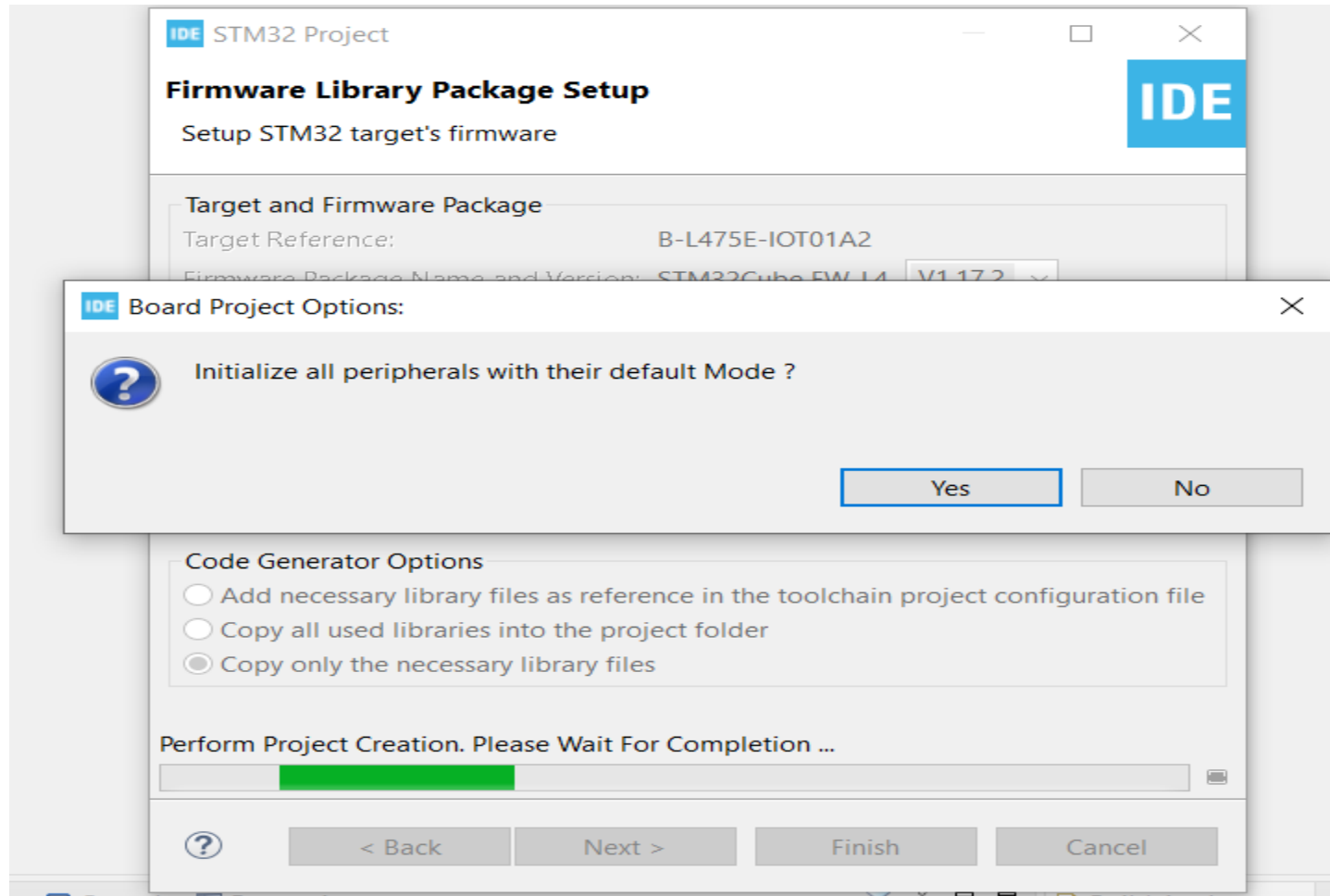
☐ Add necessary library files as reference in the toolchain project configuration file

☐ Copy all used libraries into the project folder

☒ Copy only the necessary library files

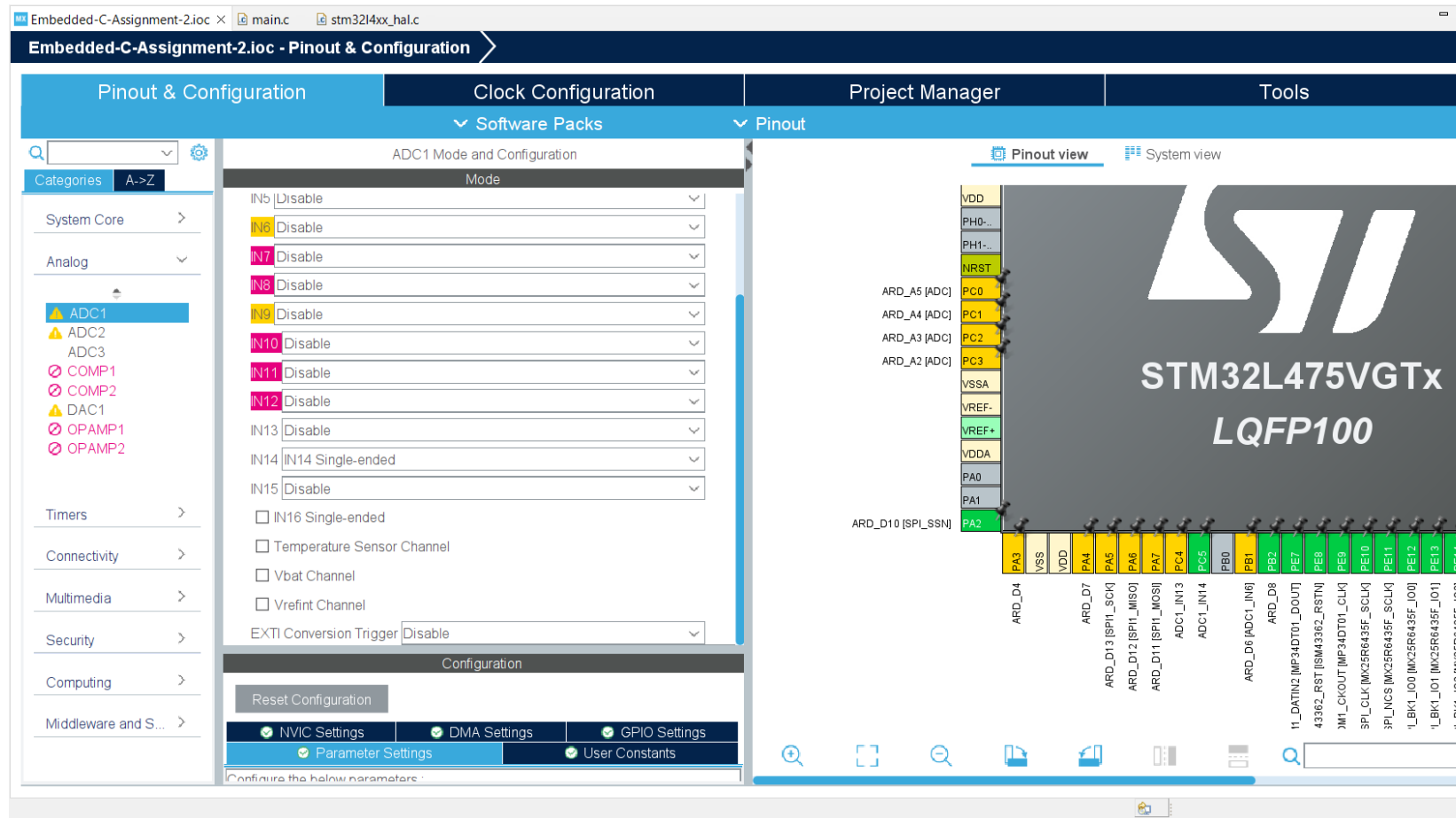
? < Back Next > Finish Cancel

Step 5. Click yes to initialize all peripherals to default





Step 7. User Story 1: Connect ARD-A0 to a 1.5VDC battery and read using ADC Polling mode. Show the output to the console

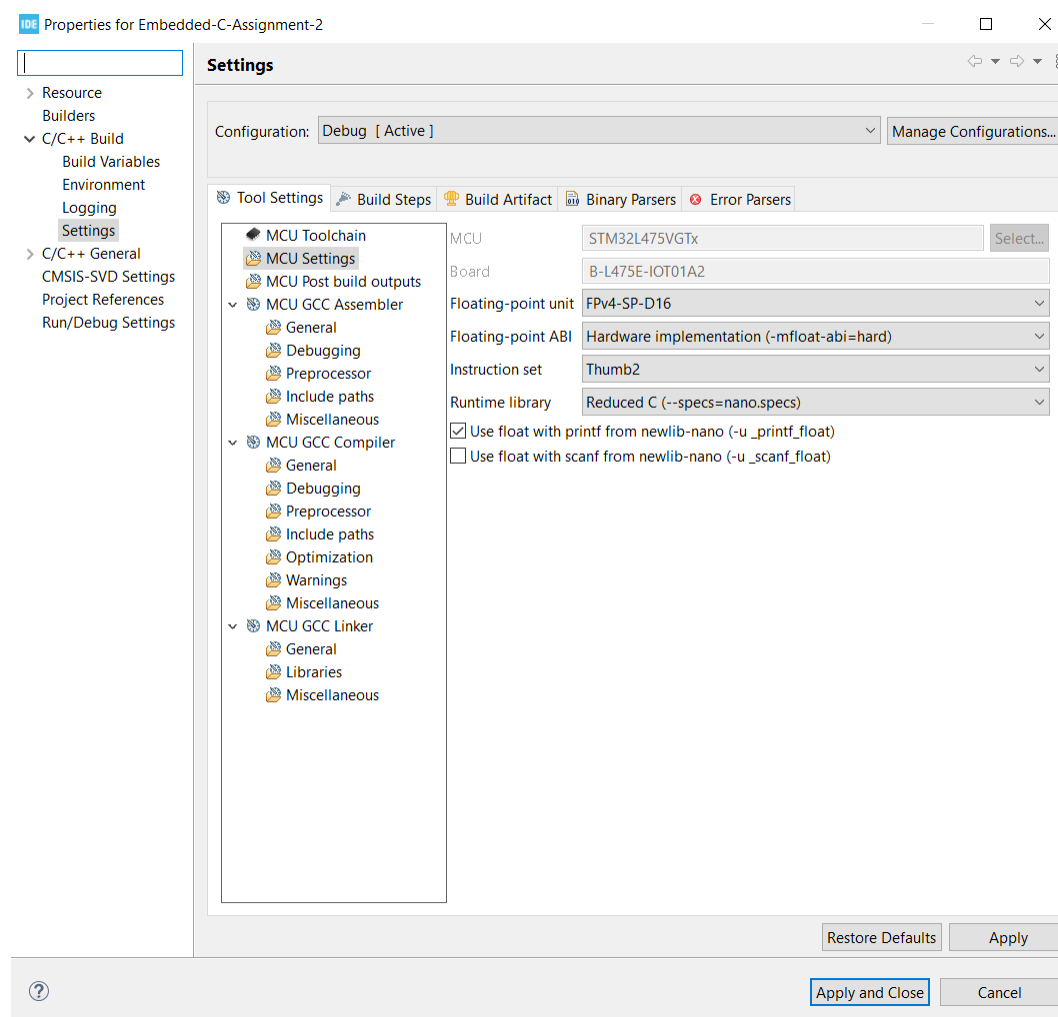




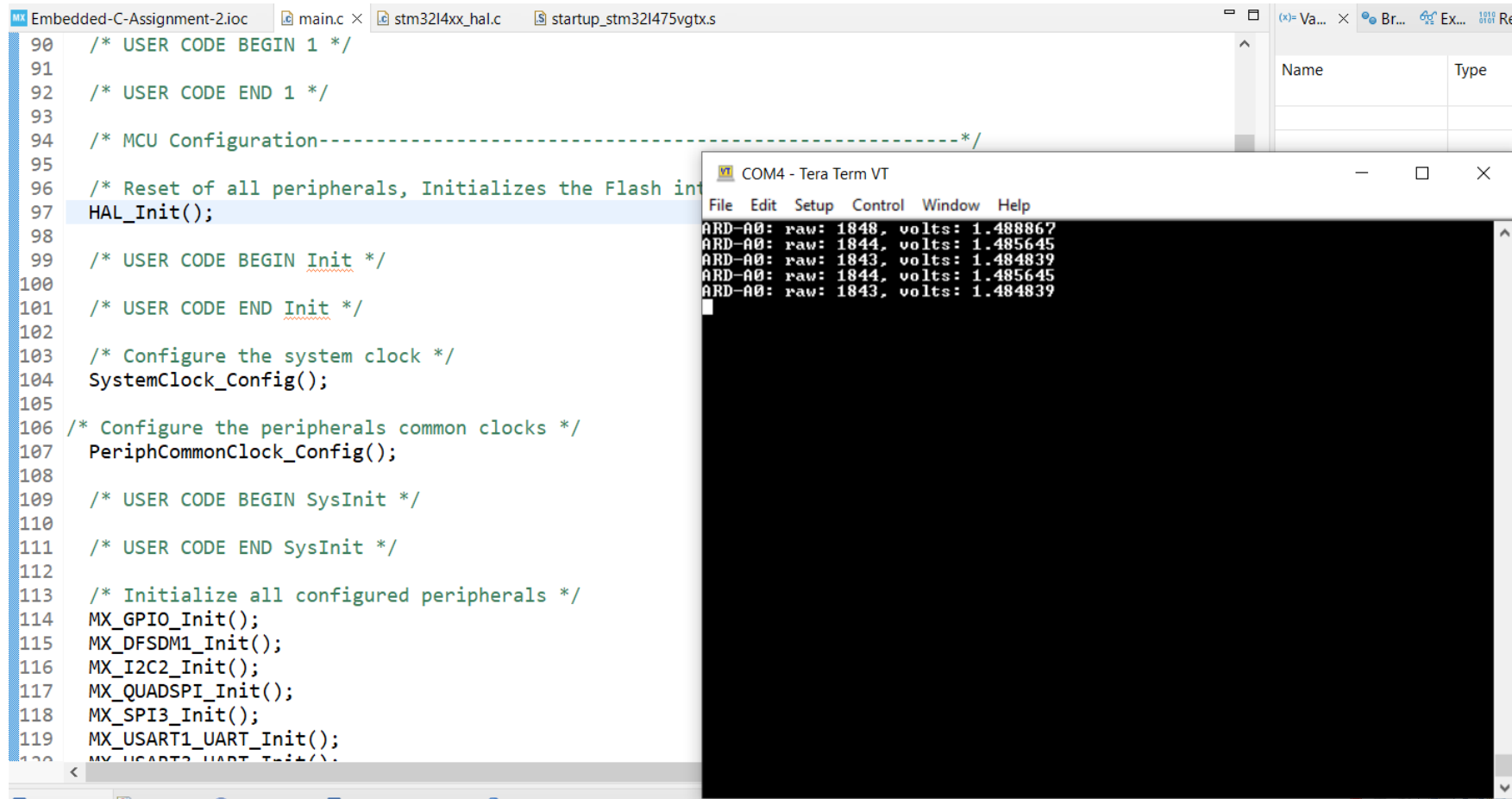
## Step 8. Write the polling mode code in main.c file

```
Embedded-C-Assignment-2.ioc  main.c ×  stm32l4xx_hal.c
122  /* USER CODE BEGIN 2 */
123  /* USER CODE BEGIN 2 */
124  HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED);
125  /* USER CODE END 2 */
126
127  /* Infinite loop */
128  /* USER CODE BEGIN WHILE */
129  while (1)
130  {
131      /* USER CODE END WHILE */
132
133      /* USER CODE BEGIN 3 */
134      HAL_GPIO_TogglePin(LED3_WIFI__LED4_BLE_GPIO_Port, LED3_WIFI__LED4_BLE_Pin);
135      HAL_Delay(1000);
136
137      // ADC start conversion
138      HAL_ADC_Start(&hadc1);
139
140      // Poll for results, timeout is 10 us
141      HAL_ADC_PollForConversion(&hadc1, 10);
142
143      uint16_t value = HAL_ADC_GetValue(&hadc1);
144      float voltage = value * (3.3 / 4096);
145
146      // Send value to console
147      char buf[100];
148      snprintf(buf, sizeof(buf), "ARD-A0: raw: %u, volts: %f\r\n", value, voltage);
149
150      HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
151
152  }
153  /* USER CODE END 3 */
154 }
```

Step 9. Since we are using floating point with printf, go to build setting and enable this flag



Step 10. Build the debug the code, open tera term and we can see the output is nearly 1.5v on the console

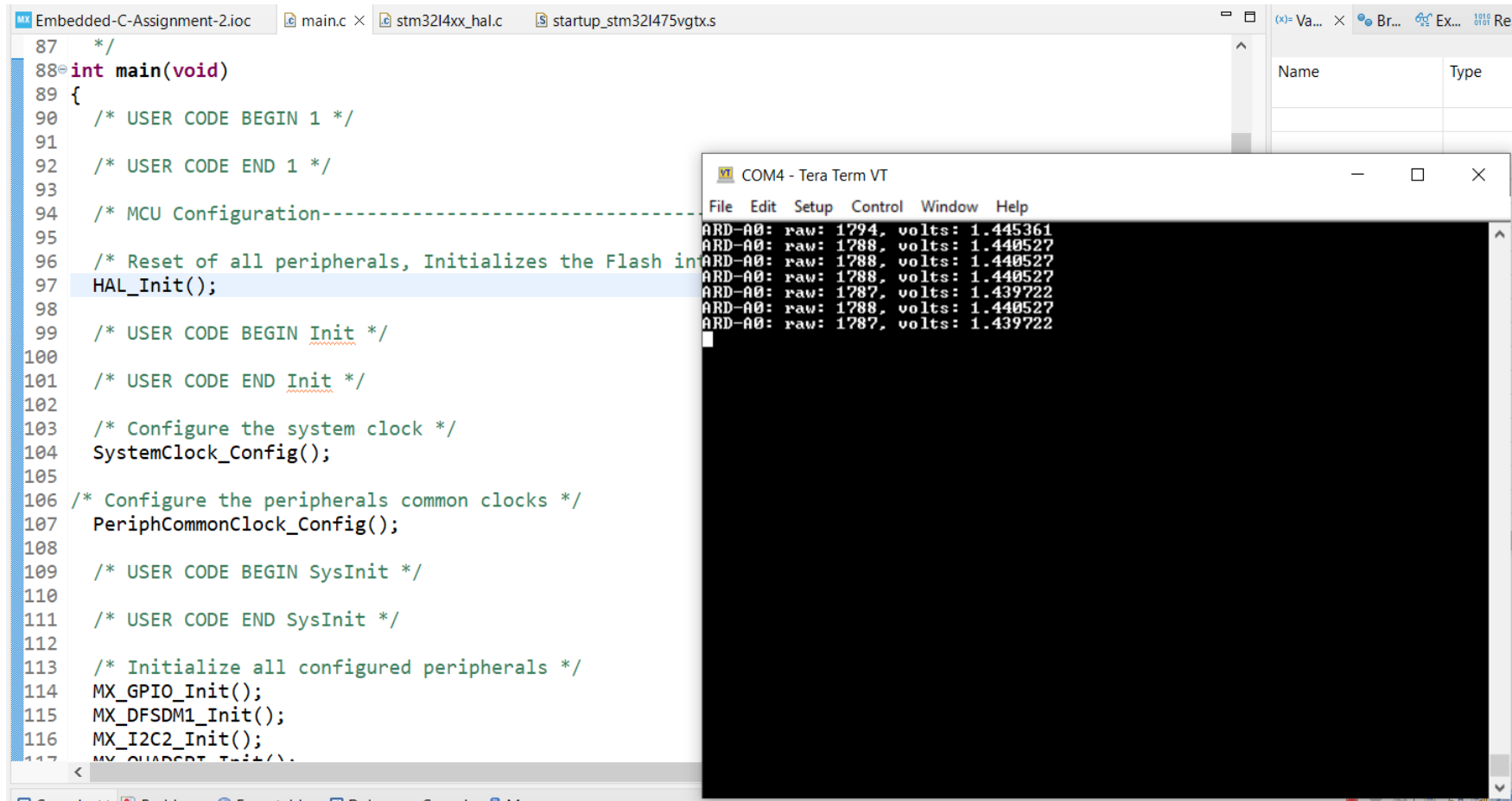


The screenshot shows an IDE with several open files: `Embedded-C-Assignment-2.ioc`, `main.c`, `stm32l4xx_hal.c`, and `startup_stm32l475vgtx.s`. The `main.c` file is active, showing C code for STM32L4 initialization. The code includes comments for user code sections and initialization functions like `HAL_Init()`, `SystemClock_Config()`, `PeriphCommonClock_Config()`, and various peripheral initialization functions (`MX_GPIO_Init()`, `MX_DFSDM1_Init()`, `MX_I2C2_Init()`, `MX_QUADSPI_Init()`, `MX_SPI3_Init()`, `MX_USART1_UART_Init()`, and `MX_USART2_UART_Init()`).

Overlaid on the IDE is a Tera Term window titled "COM4 - Tera Term VT". The window displays the output of the program, showing five lines of ADC readings from `ARD-A0`. Each line contains the raw value and the corresponding voltage in volts.

Raw Value	Volts
1848	1.488867
1844	1.485645
1843	1.484839
1844	1.485645
1843	1.484839

Step 11. Repeat user story 1 without doing calibration for user story 4. Observe that without calibration the value deviate more from 1.5V



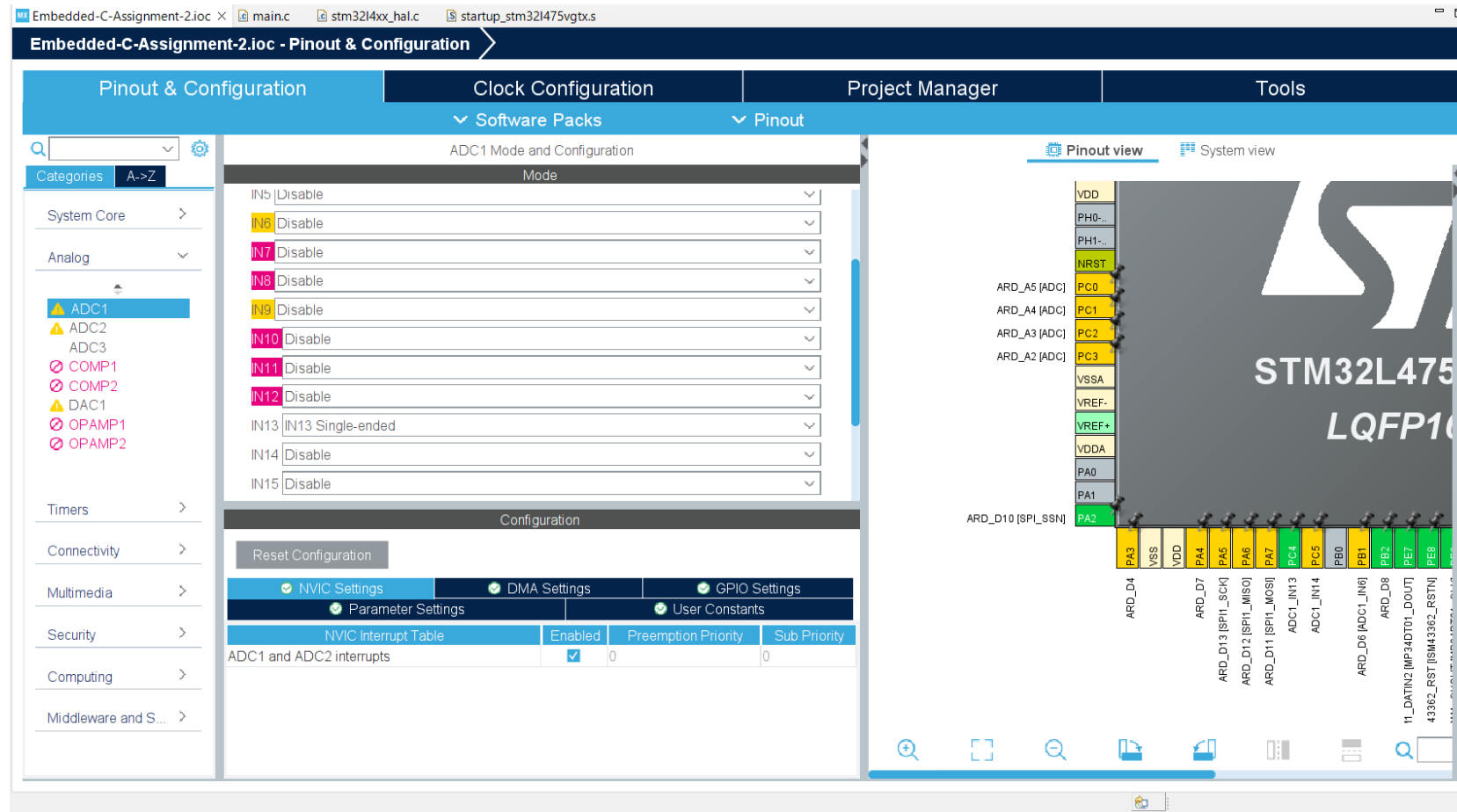
The image shows a screenshot of an IDE with two windows. The main window displays C code for an STM32 microcontroller. The code includes comments for user stories and initialization functions. The terminal window, titled 'COM4 - Tera Term VT', shows the output of the program, which consists of multiple ADC readings from ARD-A0. Each line shows a raw value and a corresponding voltage in volts. The voltage values are consistently around 1.44V, which is lower than the 1.5V mentioned in the step description.

```
87 */
88 int main(void)
89 {
90     /* USER CODE BEGIN 1 */
91
92     /* USER CODE END 1 */
93
94     /* MCU Configuration-----*/
95
96     /* Reset of all peripherals, Initializes the Flash interface and the Systick timer. */
97     HAL_Init();
98
99     /* USER CODE BEGIN Init */
100
101     /* USER CODE END Init */
102
103     /* Configure the system clock */
104     SystemClock_Config();
105
106     /* Configure the peripherals common clocks */
107     PeriphCommonClock_Config();
108
109     /* USER CODE BEGIN SysInit */
110
111     /* USER CODE END SysInit */
112
113     /* Initialize all configured peripherals */
114     MX_GPIO_Init();
115     MX_DFSDM1_Init();
116     MX_I2C2_Init();
117     MX_QUADSPI_Init();
```

COM4 - Tera Term VT

```
File Edit Setup Control Window Help
ARD-A0: raw: 1794, volts: 1.445361
ARD-A0: raw: 1788, volts: 1.440527
ARD-A0: raw: 1788, volts: 1.440527
ARD-A0: raw: 1788, volts: 1.440527
ARD-A0: raw: 1787, volts: 1.439722
ARD-A0: raw: 1788, volts: 1.440527
ARD-A0: raw: 1787, volts: 1.439722
```

Step 12. User Story 2: Connect ARD-A1 to the same 1.5VDC battery and read using ADC Interrupt mode and send the output to the console



## Step 13. Write the interrupt code in main.c file

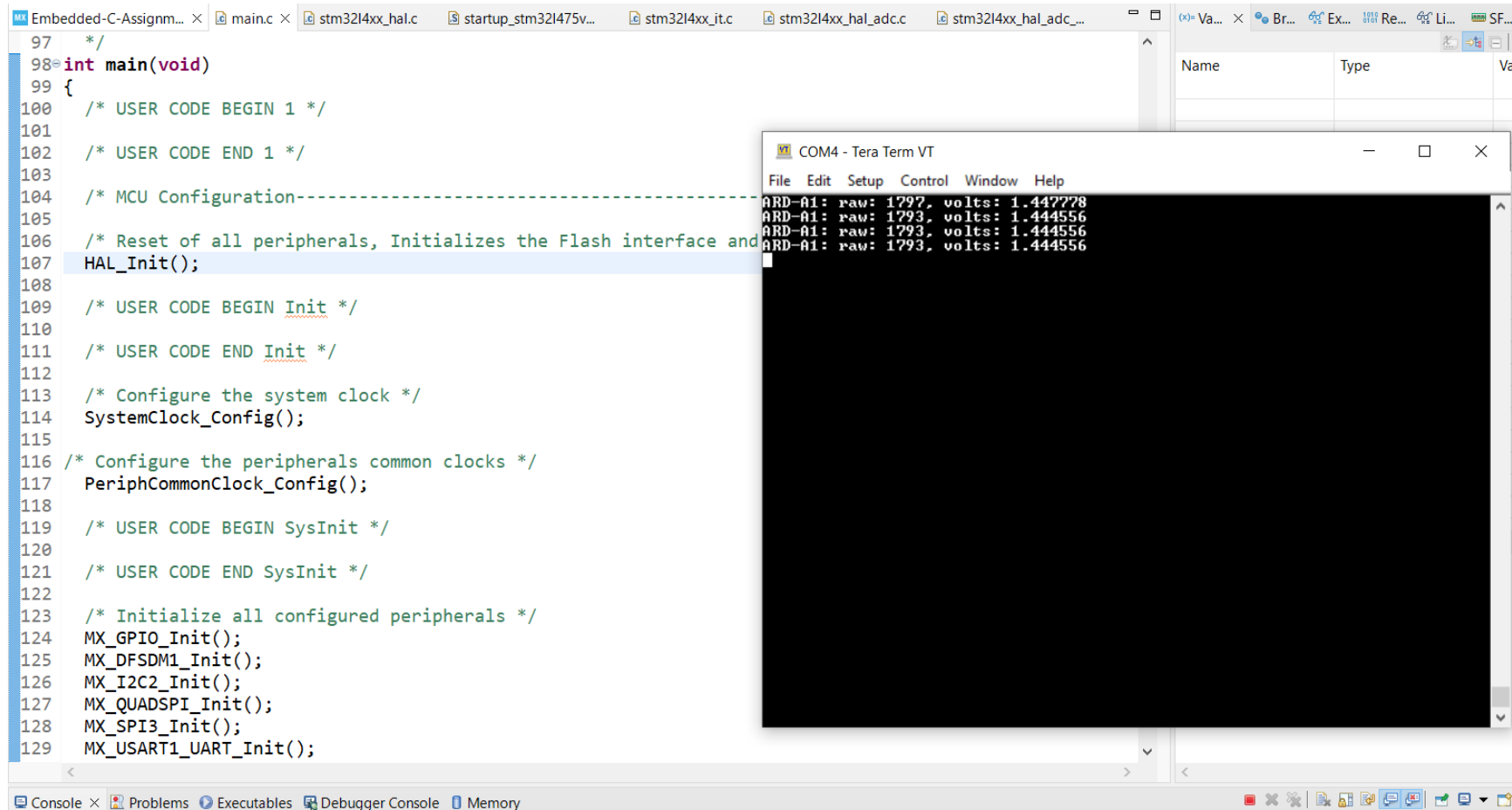
```
135 /* USER CODE END 2 */
136
137 /* Infinite loop */
138 /* USER CODE BEGIN WHILE */
139 while (1)
140 {
141     /* USER CODE END WHILE */
142
143     /* USER CODE BEGIN 3 */
144     HAL_GPIO_TogglePin(LED3_WIFI_LED4_BLE_GPIO_Port, LED3_WIFI_LED4_BLE_Pin);
145     HAL_Delay(1000);
146
147     // ADC start conversion
148     // HAL_ADC_Start(&hadc1);
149     HAL_ADC_Start_IT(&hadc1);
150
151     // Poll for results, timeout is 10 us
152     // HAL_ADC_PollForConversion(&hadc1, 10);
153     //
154     // uint16_t value = HAL_ADC_GetValue(&hadc1);
155     // float voltage = value * (3.3 / 4096);
156     //
157     // // Send value to console
158     // char buf[100];
159     // snprintf(buf, sizeof(buf), "ARD-A0: raw: %u, volts: %f\r\n", value, voltage);
160     //
161     // HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
162
163 }
164 /* USER CODE END 3 */
165 }
166
167 /**
```

```
75 /* USER CODE BEGIN PFP */
76
77 /* USER CODE END PFP */
78
79 /* Private user code -----*/
80 /* USER CODE BEGIN 0 */
81 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
82 {
83     uint16_t value = HAL_ADC_GetValue(&hadc1);
84     float voltage = value * (3.3 / 4096);
85
86     // Send value to console
87     char buf[100];
88     snprintf(buf, sizeof(buf), "ARD-A1: raw: %u, volts: %f\r\n", value, voltage);
89
90     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
91 }
92 /* USER CODE END 0 */
93
```

Step 14. Build and debug the code, observe the result on tera term

```
Embedded-C-Assignm... main.c x stm3214xx_hal.c startup_stm321475v... stm3214xx_it.c stm3214xx_hal_adc... stm3214xx_hal_adc...  
63 /* Private function prototypes -----*/  
64 void SystemClock_Config(void);  
65 void PeriphCommonClock_Config(void);  
66 static void MX_GPIO_Init(void);  
67 static void MX_DFSDM1_Init(void);  
68 static void MX_I2C2_Init(void);  
69 static void MX_QUADSPI_Init(void);  
70 static void MX_SPI3_Init(void);  
71 static void MX_USART1_UART_Init(void);  
72 static void MX_USART3_UART_Init(void);  
73 static void MX_USB_OTG_FS_PCD_Init(void);  
74 static void MX_ADC1_Init(void);  
75 /* USER CODE BEGIN PFP */  
76  
77 /* USER CODE END PFP */  
78  
79 /* Private user code -----*/  
80 /* USER CODE BEGIN 0 */  
81 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)  
82 {  
83     uint16_t value = HAL_ADC_GetValue(&hadc1);  
84     float voltage = value * (3.3 / 4096);  
85  
86     // Send value to console  
87     char buf[100];  
88     snprintf(buf, sizeof(buf), "ARD-A1: raw: %u, volts: %f\r\n", value, voltage);  
89     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);  
90 }  
91  
92 /* USER CODE END 0 */  
93  
94 /**  
95  * @brief The application entry point.
```

Step 15. Repeat user story 2, without calibration, we can see the result deviates more from 1.5V



The screenshot shows an IDE with a C code file and a terminal window. The code is for an STM32 microcontroller and includes initialization functions for HAL, system clock, peripherals, and UART. The terminal window, titled 'COM4 - Tera Term VT', displays four lines of ADC readings for channel 1, showing raw values and corresponding voltages.

```
97  */
98  int main(void)
99  {
100 /* USER CODE BEGIN 1 */
101
102 /* USER CODE END 1 */
103
104 /* MCU Configuration-----*/
105
106 /* Reset of all peripherals, Initializes the Flash interface and
107    HAL_Init();
108
109 /* USER CODE BEGIN Init */
110
111 /* USER CODE END Init */
112
113 /* Configure the system clock */
114 SystemClock_Config();
115
116 /* Configure the peripherals common clocks */
117 PeriphCommonClock_Config();
118
119 /* USER CODE BEGIN SysInit */
120
121 /* USER CODE END SysInit */
122
123 /* Initialize all configured peripherals */
124 MX_GPIO_Init();
125 MX_DFSDM1_Init();
126 MX_I2C2_Init();
127 MX_QUADSPI_Init();
128 MX_SPI3_Init();
129 MX_USART1_UART_Init();
```

COM4 - Tera Term VT

```
ARD-A1: raw: 1797, volts: 1.447778
ARD-A1: raw: 1793, volts: 1.444556
ARD-A1: raw: 1793, volts: 1.444556
ARD-A1: raw: 1793, volts: 1.444556
```



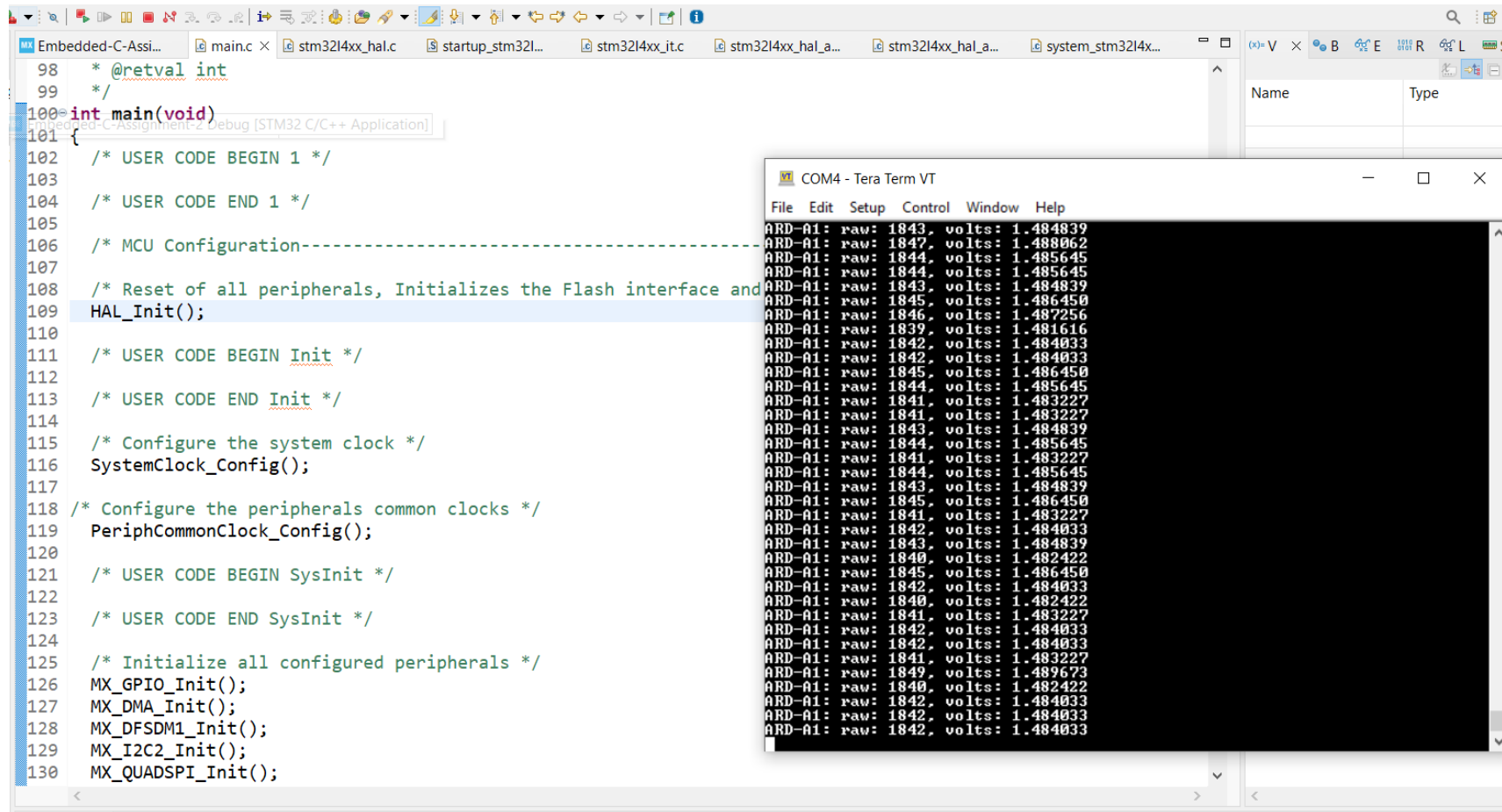


## Step 17. Write the DMA code in main.c file

```
Embedded-C-Assi... | main.c x | stm32l4xx_hal.c | startup_stm32l... | stm32l4xx_it.c | stm32l4xx_hal_a... | stm32l4xx_hal_a... | system_stm32l4x...
135 MX_ADC1_Init();
136 /* USER CODE BEGIN 2 */
137 HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED);
138 /* USER CODE END 2 */
139
140 /* Infinite loop */
141 /* USER CODE BEGIN WHILE */
142 while (1)
143 {
144     /* USER CODE END WHILE */
145
146     /* USER CODE BEGIN 3 */
147     HAL_GPIO_TogglePin(LED3_WIFI_LED4_BLE_GPIO_Port, LED3_WIFI_LED4_BLE_Pin);
148     HAL_Delay(1000);
149
150     // ADC start conversion
151 // HAL_ADC_Start(&hadc1);
152 // HAL_ADC_Start_IT(&hadc1);
153 HAL_ADC_Start_DMA(&hadc1, (uint32_t*)&value, 1);
154
155     // Poll for results, timeout is 10 us
156 // HAL_ADC_PollForConversion(&hadc1, 10);
157 //
158 // uint16_t value = HAL_ADC_GetValue(&hadc1);
159 // float voltage = value * (3.3 / 4096);
160 //
161 // // Send value to console
162 // char buf[100];
163 // snprintf(buf, sizeof(buf), "ARD-A0: raw: %u, volts: %f\r\n", value, voltage);
164 //
165 // HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
166
167 }
```

```
--
81 // Private user code -----*/
82 /* USER CODE BEGIN 0 */
83 static uint16_t value;
84
85 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
86 {
87     float voltage = value * (3.3 / 4096);
88     // Send value to console
89     char buf[100];
90     snprintf(buf, sizeof(buf), "ARD-A1: raw: %u, volts: %f\r\n", value, voltage);
91
92     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
93 }
94 /* USER CODE END 0 */
95
```

Step 18, Build and debug the code, observe the result on tera term



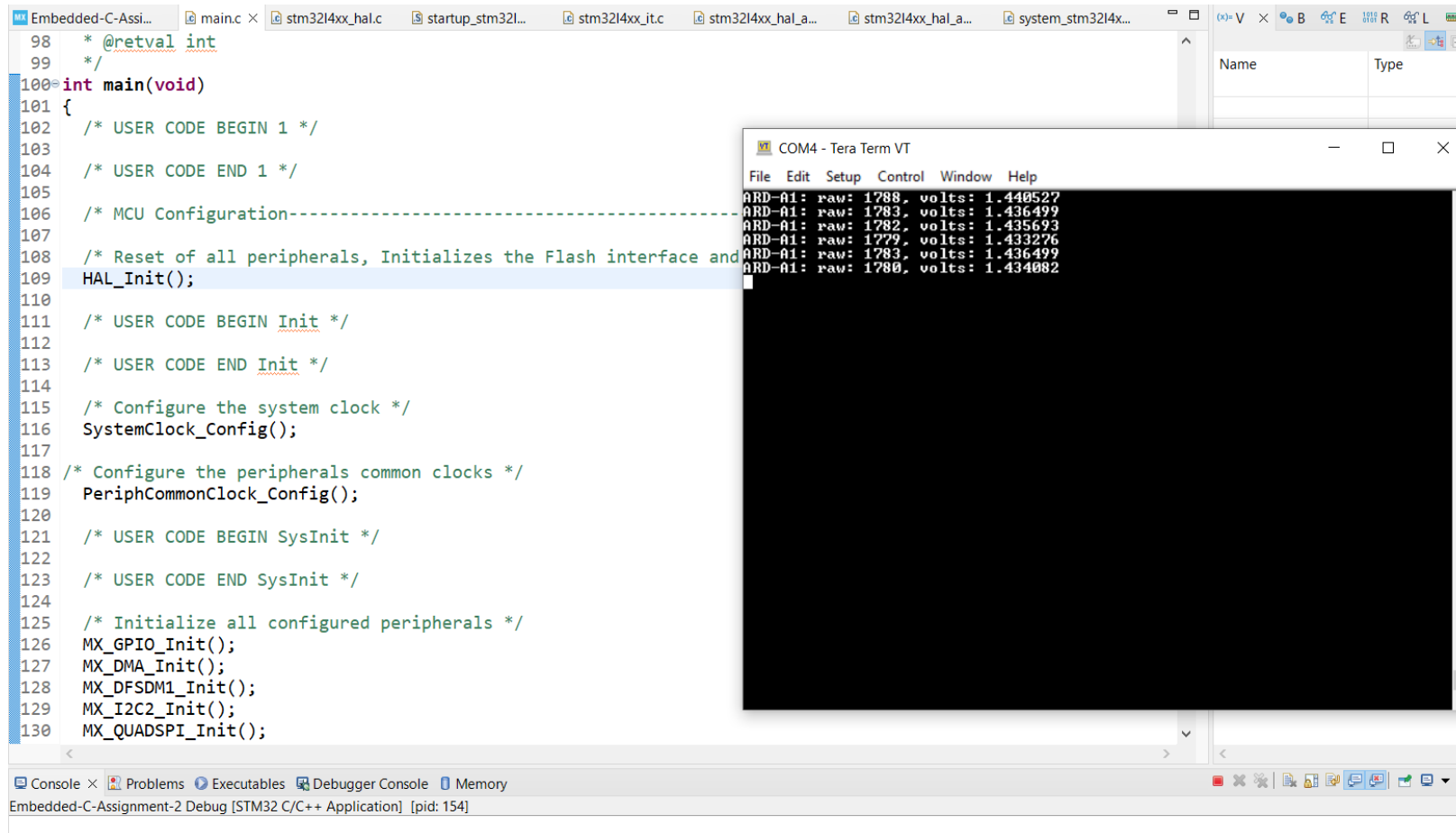
The image shows a screenshot of an IDE (likely Keil uVision) with a C program for STM32. The code is in `main.c` and includes comments for user code and initialization. The code is as follows:

```
98  * @retval int
99  */
100 int main(void)
101 {
102     /* USER CODE BEGIN 1 */
103
104     /* USER CODE END 1 */
105
106     /* MCU Configuration-----*/
107
108     /* Reset of all peripherals, Initializes the Flash interface and
109     HAL_Init();
110
111     /* USER CODE BEGIN Init */
112
113     /* USER CODE END Init */
114
115     /* Configure the system clock */
116     SystemClock_Config();
117
118     /* Configure the peripherals common clocks */
119     PeriphCommonClock_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_DMA_Init();
128     MX_DFSDM1_Init();
129     MX_I2C2_Init();
130     MX_QUADSPI_Init();
```

Overlaid on the IDE is a terminal window titled "COM4 - Tera Term VT". It displays a series of ADC readings from an ARD-A1 sensor. Each line shows the raw value and the corresponding voltage in volts. The readings are as follows:

Raw Value	Volts
1843	1.484839
1847	1.488062
1844	1.485645
1844	1.485645
1843	1.484839
1845	1.486450
1846	1.487256
1839	1.481616
1842	1.484033
1842	1.484033
1845	1.486450
1844	1.485645
1841	1.483227
1841	1.483227
1843	1.484839
1844	1.485645
1841	1.483227
1844	1.485645
1843	1.484839
1845	1.486450
1841	1.483227
1842	1.484033
1843	1.484839
1840	1.482422
1845	1.486450
1842	1.484033
1840	1.482422
1841	1.483227
1842	1.484033
1842	1.484033
1841	1.483227
1849	1.489673
1840	1.482422
1842	1.484033
1842	1.484033

Step 19. Repeat user story 3, without calibration, we can see the result deviates more from 1.5V



The screenshot shows an IDE with a C file named `main.c` and a terminal window titled "COM4 - Tera Term VT".

**Code in `main.c`:**

```
98  * @retval int
99  */
100 int main(void)
101 {
102     /* USER CODE BEGIN 1 */
103
104     /* USER CODE END 1 */
105
106     /* MCU Configuration-----*/
107
108     /* Reset of all peripherals, Initializes the Flash interface and
109     HAL_Init();
110
111     /* USER CODE BEGIN Init */
112
113     /* USER CODE END Init */
114
115     /* Configure the system clock */
116     SystemClock_Config();
117
118     /* Configure the peripherals common clocks */
119     PeriphCommonClock_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_DMA_Init();
128     MX_DFSDM1_Init();
129     MX_I2C2_Init();
130     MX_QUADSPI_Init();
```

**Terminal Output (COM4 - Tera Term VT):**

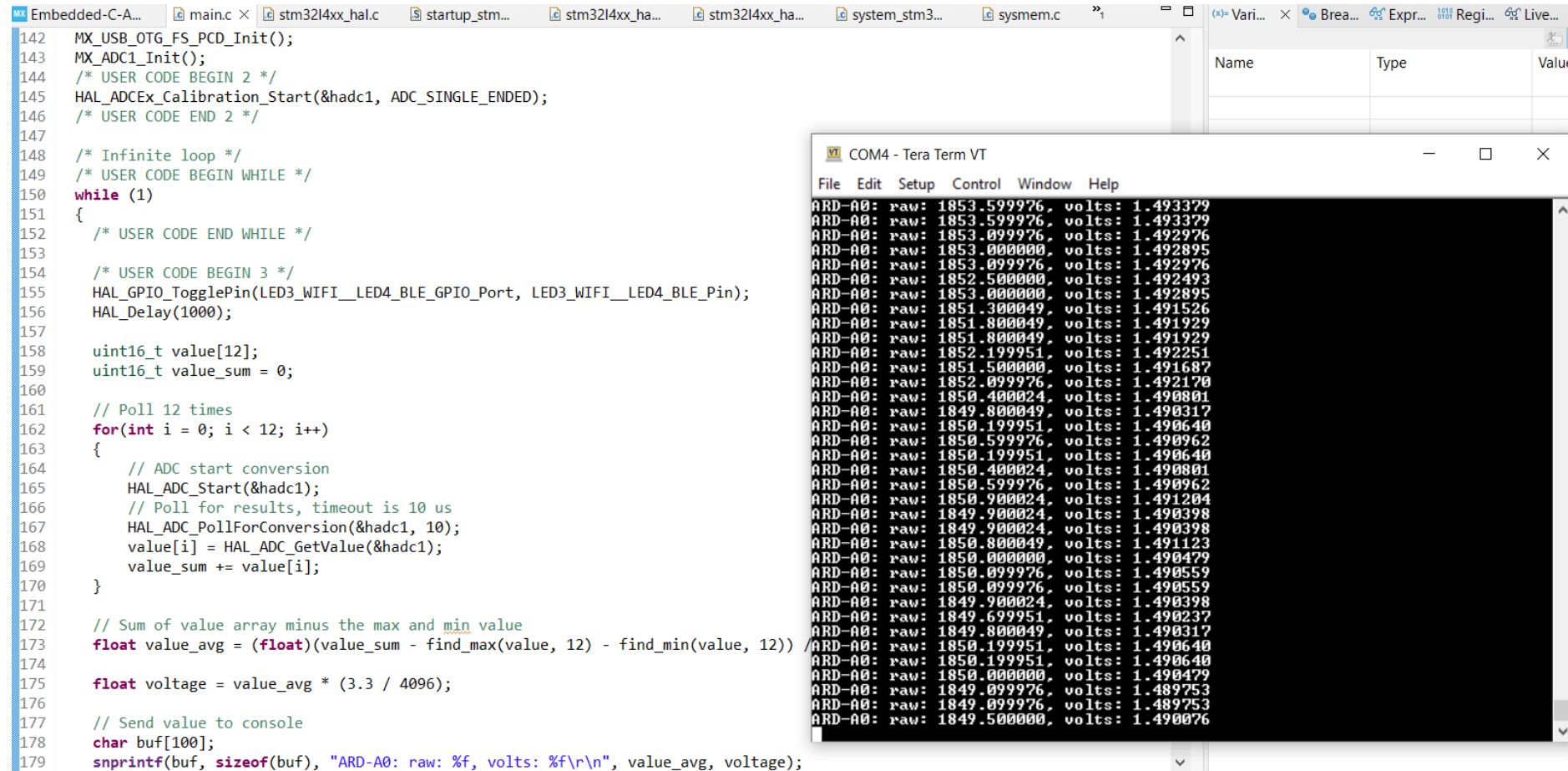
```
ARD-A1: raw: 1788, volts: 1.440527
ARD-A1: raw: 1783, volts: 1.436499
ARD-A1: raw: 1782, volts: 1.435693
ARD-A1: raw: 1779, volts: 1.433276
ARD-A1: raw: 1783, volts: 1.436499
ARD-A1: raw: 1780, volts: 1.434082
```

Step 20. User Store 5: Only for ADC Polling Mode, when it comes time to read, instead of reading a single time, read the ADC 12 times "back-to-back", dropping the smallest value and also the largest value

```
145 HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED);
146 /* USER CODE END 2 */
147
148 /* Infinite loop */
149 /* USER CODE BEGIN WHILE */
150 while (1)
151 {
152     /* USER CODE END WHILE */
153
154     /* USER CODE BEGIN 3 */
155     HAL_GPIO_TogglePin(LED3_WIFI_LED4_BLE_GPIO_Port, LED3_WIFI_LED4_BLE_Pin);
156     HAL_Delay(1000);
157
158     uint16_t value[12];
159     uint16_t value_sum = 0;
160
161     // Poll 12 times
162     for(int i = 0; i < 12; i++)
163     {
164         // ADC start conversion
165         HAL_ADC_Start(&hadc1);
166         // Poll for results, timeout is 10 us
167         HAL_ADC_PollForConversion(&hadc1, 10);
168         value[i] = HAL_ADC_GetValue(&hadc1);
169         value_sum += value[i];
170     }
171
172     // Sum of value array minus the max and min value
173     float value_avg = (float)(value_sum - find_max(value, 12) - find_min(value, 12)) / 10;
174
175     float voltage = value_avg * (3.3 / 4096);
176
177     // Send value to console
178     char buf[100];
179     snprintf(buf, sizeof(buf), "ARD-A0: raw: %f, volts: %f\r\n", value_avg, voltage);
180
181     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
182
183 }
184 /* USER CODE END 3 */
185 }
186
```

```
79 /* Private user code -----
80 /* USER CODE BEGIN 0 */
81 uint16_t find_max(uint16_t arr[], uint8_t size)
82 {
83     int max = arr[0];
84     for(int i = 0; i < size; i++){
85         if(arr[i] > max){
86             max = arr[i];
87         }
88     }
89     return max;
90 }
91
92 uint16_t find_min(uint16_t arr[], uint8_t size)
93 {
94     int min = arr[0];
95     for(int i = 0; i < size; i++){
96         if(arr[i] < min){
97             min = arr[i];
98         }
99     }
100     return min;
101 }
102
103 /* USER CODE END 0 */
104
```

Step 21. User Store 6: Compare and contrast your output from User Story 1. I found out that the averaging method has more precise value compares to User Story 1, and the value does not jump around much.



The image shows a screenshot of an IDE (likely STM32CubeIDE) with a C source file open. The code implements an infinite loop that initializes an ADC, configures a GPIO pin, and then enters a while loop. Inside the while loop, it polls the ADC 12 times, calculates the average of the readings (excluding the maximum and minimum values), and then calculates the voltage using a formula:  $\text{voltage} = \text{value\_avg} * (3.3 / 4096)$ . The code also includes a comment about the averaging method being more precise.

```
142 MX_USB_OTG_FS_PCD_Init();
143 MX_ADC1_Init();
144 /* USER CODE BEGIN 2 */
145 HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED);
146 /* USER CODE END 2 */
147
148 /* Infinite loop */
149 /* USER CODE BEGIN WHILE */
150 while (1)
151 {
152     /* USER CODE END WHILE */
153
154     /* USER CODE BEGIN 3 */
155     HAL_GPIO_TogglePin(LED3_WIFI__LED4_BLE_GPIO_Port, LED3_WIFI__LED4_BLE_Pin);
156     HAL_Delay(1000);
157
158     uint16_t value[12];
159     uint16_t value_sum = 0;
160
161     // Poll 12 times
162     for(int i = 0; i < 12; i++)
163     {
164         // ADC start conversion
165         HAL_ADC_Start(&hadc1);
166         // Poll for results, timeout is 10 us
167         HAL_ADC_PollForConversion(&hadc1, 10);
168         value[i] = HAL_ADC_GetValue(&hadc1);
169         value_sum += value[i];
170     }
171
172     // Sum of value array minus the max and min value
173     float value_avg = (float)(value_sum - find_max(value, 12) - find_min(value, 12)) / 10;
174
175     float voltage = value_avg * (3.3 / 4096);
176
177     // Send value to console
178     char buf[100];
179     snprintf(buf, sizeof(buf), "ARD-A0: raw: %f, volts: %f\r\n", value_avg, voltage);
```

The terminal window (COM4 - Tera Term VT) displays the output of the program, showing raw ADC values and calculated voltages. The output is as follows:

Raw Value	Volts
1853.599976	1.493379
1853.599976	1.493379
1853.099976	1.492976
1853.000000	1.492895
1853.099976	1.492976
1852.500000	1.492493
1853.000000	1.492895
1851.300049	1.491526
1851.800049	1.491929
1851.800049	1.491929
1852.199951	1.492251
1851.500000	1.491687
1852.099976	1.492170
1850.400024	1.490801
1849.800049	1.490317
1850.199951	1.490640
1850.599976	1.490962
1850.199951	1.490640
1850.400024	1.490801
1850.599976	1.490962
1850.900024	1.491204
1849.900024	1.490398
1849.900024	1.490398
1850.800049	1.491123
1850.000000	1.490479
1850.099976	1.490559
1850.099976	1.490559
1849.900024	1.490398
1849.699951	1.490237
1849.800049	1.490317
1850.199951	1.490640
1850.199951	1.490640
1850.000000	1.490479
1849.099976	1.489753
1849.099976	1.489753
1849.500000	1.490076



## Appendix. Hardware connections

